UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Integración de emuladores en tiempo real sobre plataforma Zynq

Autor: Javier J. Chavarino Martínez

Tutor: Fernando Jesús López Colino
Ponente: Ángel de Castro Martín

Junio 2016

Integración de emuladores en tiempo real sobre plataforma Zynq

AUTOR: Javier Jesús Chavarino Martínez
TUTOR: Fernando Jesús López Colino

Hardware and Control Technology Laboratory
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

Resumen

Este TFG (Trabajo de Fin de Grado) consiste en desarrollar una aplicación que permita realizar emulaciones de dispositivos hardware de forma remota en una FPGA. El resultado obtenido es una aplicación web, diseñada bajo la arquitectura Modelo-Vista-Controlador, con cinco páginas web, a través de las cuales se permite realizar una serie de funcionalidades específicas que permiten al usuario llevar acabo los objetivos marcados por el TFG. Finalmente, a través de una serie de pruebas se comprueba el correcto funcionamiento del sistema completo.

Palabras clave

FPGA, System on Chip, emulación de sistemas, sistema web, Modelo-Vista-Controlador.

Abstract

This TFG (Final degree Project) consist in the developing of an application that allows to do emulations of hardware devices remote-controlled with a FPGA. The result obtained is a web application, designed under the Model-View-Controller architecture, with five web pages, through which it is allowed to do some specific functionalities. Through this web application, the user can get the set targets in the TFG. Finally, through several test, it is possible to check the proper functioning of the complete system.

Keywords

FPGA, System on Chip, system emulation, web system, Model-View-Controller.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

Los circuitos integrados han revolucionado el mundo de la electrónica. Estos forman una importantísima parte de nuestra vida cotidiana, ya que son usados prácticamente para todas las aplicaciones electrónicas existentes, como por ejemplo ordenadores, teléfonos móviles, o lavadoras.

En sus orígenes, los circuitos integrados eran caros y complicados de diseñar. Además, sus aplicaciones tenían una funcionalidad rígida y específica, imposibilitando la escalabilidad y mantenimiento.

A lo largo del tiempo, con el objetivo de facilitar y agilizar los procesos de desarrollo de circuitos integrados, la industria ha desarrollado diversas tecnologías, como por ejemplo la FPGA (Field Programable Gate Array)[1].

Las FPGAs, son dispositivos que permiten ser programados, a partir de un lenguaje específico de programación descriptiva, con aplicaciones, emulando así circuitos integrados. En ellas se pueden emular desde pequeñas aplicaciones (puertas lógicas), hasta complejos dispositivos (Procesadores). Con respecto a los circuitos integrados físicos, las FPGAs son mucho más flexibles y baratas, pero presentan algunos inconvenientes (son más lentos, menos potentes).

Paralelamente, y a lo largo de los años, con su gran avance y popularidad, todos los sistemas informáticos han ido proyectándose de cara a internet, la Web y los sistemas distribuidos [2]. Esto se debe a muchas de las ventajas que aportan a las aplicaciones, y como puede evidenciarse, las herramientas de implementaciones hardware, lo han aprovechado, y se han complementado, incorporando algunos de sus aspectos más positivos, como puede ser el acceso remoto.

1.1 Motivación

Este Trabajo de Fin de Grado plantea resolver:

- La emulación de diseños hardware implementados en un lenguaje de programación descriptivo, típicamente VHDL, en una FPGA.

- Además, dicha emulación debe permitir de forma remota modificar los parámetros de estos diseños y observar el comportamiento de sus resultados.

1.2 Objetivos TFG

El objetivo de este proyecto es establecer un marco de trabajo que permita por un lado emular diseños hardware en una FPGA, y por otro, mediante tecnología web poder editar las entradas parametrizadas de los diseño y observar sus diversos resultados, implementando finalmente una aplicación de emulación hardware sobre FPGA remoto. A parte, también se marcan objetivos secundarios como realizar un sistema usable, accesible y fácil de mantener.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos: "Introducción", "Estado del arte", "Diseño y desarrollo", "Integración, pruebas y resultados", "Conclusión" y "Trabajo futuro". En la "Introducción" se explicará una introducción a la motivación y a los objetivos del proyecto. En el capítulo "Estado del arte" se analizaran los trabajos y proyectos con soluciones alternativas al proyecto implementado. En el de "Diseño y desarrollo" se desarrollará la idea del proyecto a través del diseño y desarrollo de las distintas partes del proyecto. El capítulo de "Integración, pruebas y resultados" se explicará al lector la integración, pruebas y resultados de las diferentes partes del proyecto. Finalmente, con las "Conclusiones y trabajo futuro" se expondrá el resultado del proyecto desarrollado y las posibilidades de mejora en el futuro.

2 Estado del arte

A lo largo de este apartado se van a analizar proyectos y tecnologías existentes que trabajan con FPGAs, al igual que el trabajo propuesto.

2.1 Laboratorio virtual para la programación de FPGAs

Es un marco de trabajo implementado por la Universidad de Córdoba en 2005, es una aplicación web orientada a la docencia virtual, con el objetivo de aportar a los alumnos una herramienta práctica para programar FPGA's, de forma remota, a través de Internet y para una asignatura concreta [3]. El esquema de esta aplicación puede verse en la figura 2-1

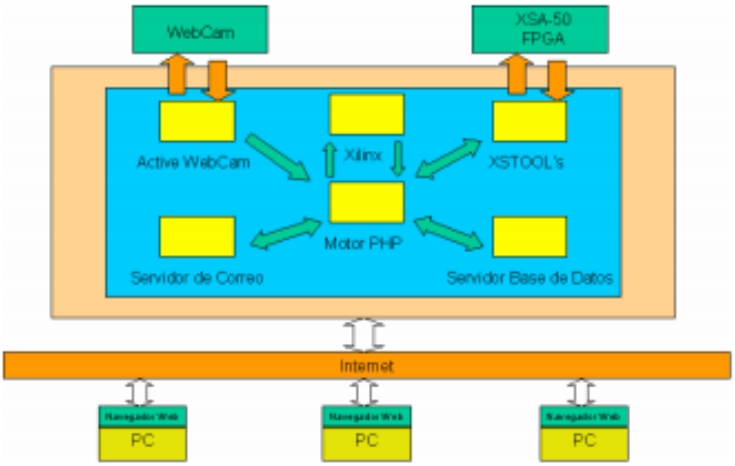


Figura 2-1 Laboratorio virtual para programar FPGA – Diseño

Esta aplicación web proporciona su funcionalidad a través de páginas html y permite de forma remota:

- Registro y autenticación en la plataforma.
- Subir y descargar ficheros VHDL del servidor.
- Compilar ficheros VHDL y cargarlos en la FPGA.
- Modificar parámetros y comprobar resultados de la FPGA.
- Ver el comportamiento de la FPGA a través de una webcam instalada en el sistema.

Análisis

Este proyecto está orientado a un ámbito docente. Soporta, en la misma plataforma, todos los pasos de desarrollo de un dispositivo (diseño, compilación, carga), además de permitir probar los dispositivos.

Las páginas del proceso de subir, compilar y cargar son muy intuitivas.

La página de visualización e interacción con los registros de la FPGA, es muy específica para la FPGA implantada en el sistema, por lo que es poco escalable a otros dispositivos. Esta página también es bastante técnica, por lo que para usarla se debe tener conocimiento de alto nivel sobre la FPGA para poder usarla Este sistema permite accesos simultáneos a la misma plataforma, pero sólo permite un acceso a la vez a la zona de la FPGA.

2.2 Quartus II

Es una herramienta de software desarrollada por Altera para realizar tanto análisis como síntesis de diseños realizados en lenguaje descriptivo HDL[4]. Permite editar diseños hardware a través de una aplicación web de forma gratuita.

Quartus II, aparte de dar soporte para el diseño y compilación de los elementos hardware, también da soporte para configurar la FPGA de destino, con el objetivo de poder probar los diseños implementados.

Adicionalmente, a través de una de sus herramientas, da la posibilidad de realizar una programación remota de la FPGA. La herramienta se llama JTAG Server[5]. JTAG Server permite acceder a los cables JTAG conectados al ordenador remoto, en el cual este elemento software está instalado. La/s FPGA/s se encuentran conectadas a estos cables JTAG, permitiendo que puedan programarse de forma remota.

El cliente podrá acceder al ordenador remoto a través una herramienta gráfica también proporcionada por Quartus II. Esta herramienta proporciona funcionalidad básica de acceso a servidor, elección de FPGA e interfaz de carga de diseños.

Análisis

Esta herramienta da soporte para diseño, compilación y carga de dispositivos. Para usar el modo remoto, el ordenador servidor debe estar permanentemente activo y encendido, sólo para dar soporte a pequeñas peticiones que haga el cliente sobre la FPGA.

2.3 Remote Soc/FPGA Plataform Configuration for Cloud Applications

Es un proyecto desarrollado por el departamento de electrónica de la universidad de Transilvania, Rumanía, junto a la empresa "Digital Optics Corporation", Rumania.

Este proyecto es una implementación de un sistema remoto FPGA basado en una una tecnología SoC(Sytem on Chip) [6], en la cual se ha implementado un sistema Web por Internet, que permite a través de una GUI(Graphical User Interface)[7], de páginas JSP (Java Servlet Page)[8], realizar acciones determinadas sobre la FPGA remota[9].

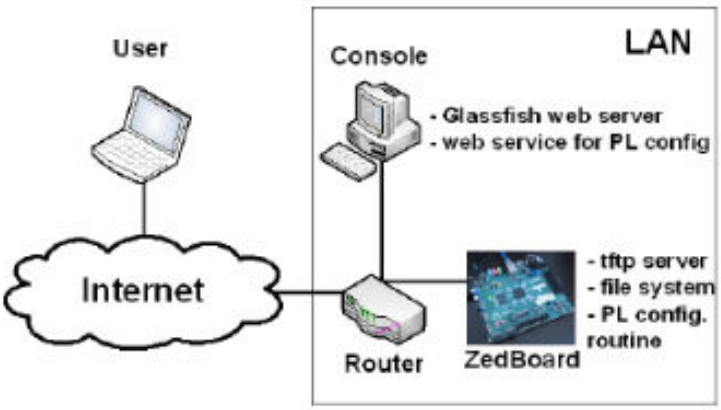


Figura 2-2 Remote SoC/FPGA Applications - diseño

La Figura 2-2 muestra el diseño de la arquitectura del sistema. Esta consta de tres elementos:

Un dispositivo ZedBoard[10], que es el sistema SoC que contiene la FPGA, en la cual se alojaran finalmente todos los diseños Hardware implementados. Este dispositivo se comunica con el resto de elementos a través de internet.

El segundo elemento, es un ordenador en modo servidor con Glassfish.[11] Este proporciona a los clientes de todos los servicios Web y los provee de páginas JSP, con el objetivo de facilitar la interacción con el sistema. Los servicios que puede proporcionar a través de su GUI son: subir un nuevo fichero de configuración, elegir la configuración a cargar en la FPGA, borrar configuración y para ver el comportamiento de los diseños implementados en la FPGA.

El último elemento, se trata del medio de transporte que comunica los dos elementos anteriores entre ellos y con las peticiones del exterior. Esto es una red LAN interconectada mediante un router.

Análisis
Es un sistema remoto que a través una web permite emular diseños hardware. Para que el sistema funcione, necesita que varios elementos estén activos, simultáneamente, el ordenador y el SoC. Todos los recursos de un ordenador sólo para atender peticiones exteriores. Todos los del SoC, sólo para correr los diseños hardware. Por lo tanto, se desaprovechan recursos. Permite tener subidas múltiples configuraciones de la FPGA a la vez. Sólo una a la vez.

2.4 Conclusión

Después analizar múltiples proyectos de un ámbito relacionado con la programación de las FPGAS, se puede llegar a la conclusión de que, aun siendo muy diferentes los proyectos entre sí, suelen tener un objetivo común: conseguir emular diseños hardware de forma remota. Además, casi la gran mayoría de los proyectos encontrados, siguen un patrón similar para llevar a cabo su objetivo de FPGA remota: un sistema web que permite la interacción con dicha FPGA.

3 Diseño y desarrollo

En este capítulo se explicará cómo se ha desarrollado el proyecto: requisitos, arquitectura del sistema implementado, las tecnologías utilizadas, y finalmente, el diseño y desarrollo de los elementos que lo componen.

3.1 Requisitos

Debido a los objetivos marcados en el proyecto, éste proporciona de forma directa una serie de requisitos funcionales y no funcionales de forma directa.

Requisitos funcionales

- Posibilidad de enviar ficheros de configuración / reprogramación de la FPGA
- Posibilidad de cargar dichos archivos de configuración / programación en la FPGA
- Posibilidad de dar a conocer al sistema los dispositivos alojados en la FPGA
- Posibilidad de quitar del sistema los dispositivos dados a conocer
- Posibilidad de ver los resultados de las parametrizaciones de los dispositivos

Requisitos no funcionales

- Arquitectura hardware de procesamiento y FPGA para albergar las implementaciones hardware.
- Servidor web para recibir las peticiones remotas
- Base de datos
- Interfaz Gráfica de usuario

3.2 Arquitectura del sistema

Después de identificar los requisitos necesarios para nuestro sistema, entonces se procede a realizar la arquitectura del sistema. A continuación, en la Figura 3-1, el lector puede observar dicha arquitectura.

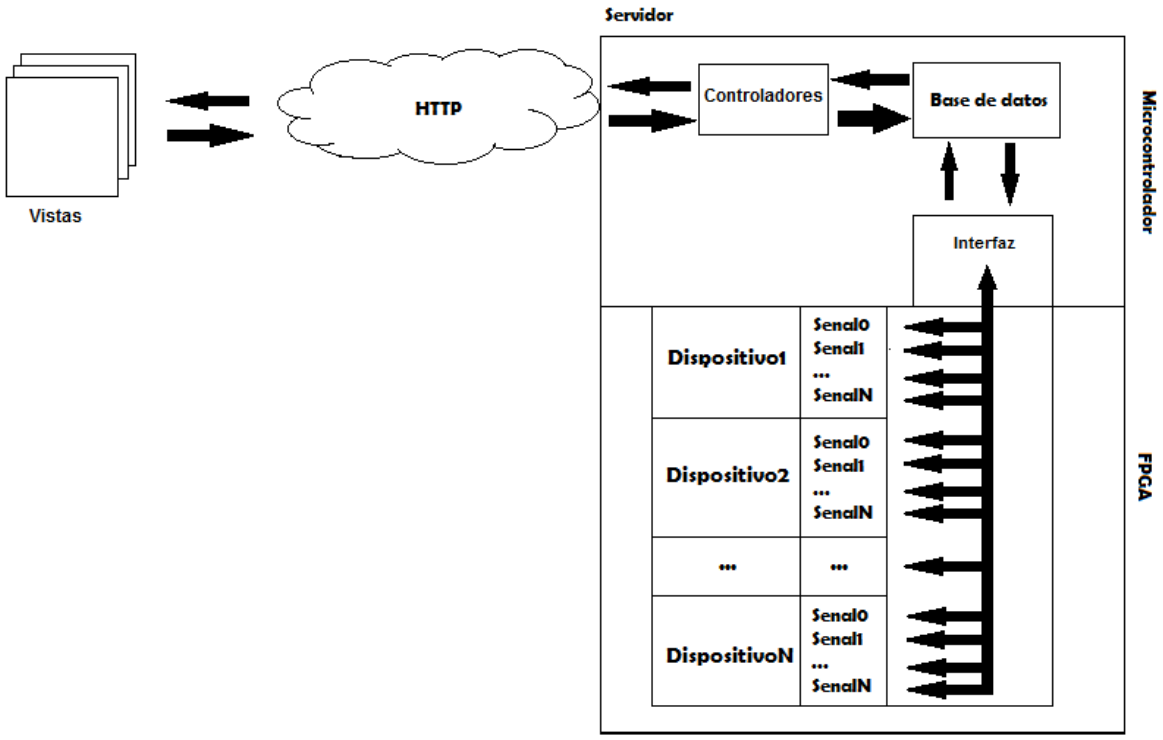


Figura 3-1 Diseño de arquitectura del sistema.

Como puede verse en la Figura 3-1, la arquitectura propuesta consta de dos partes: el microcontrolador y la de FPGA. En el microcontrolador se aloja todo el trabajo de los elementos necesarios para poder llevar a cabo la comunicación entre el usuario y la FPGA. La parte de FPGA alojará únicamente los diseños hardware implementados.

Microcontrolador

En el microcontrolador se ejecuta un proceso sigue el diseño de la arquitectura MVC[12] (Modelo-Vista-Controlador), con el objetivo de dotar a éste un diseño basado en capas. Cada capa tiene una funcionalidad concreta, y cada una de ellas trabaja de forma independiente. Las capas y elementos resultantes del diseño son las siguientes:

- Vistas: única capa con la que el usuario podrá interactuar. En este proyecto, esta será la capa donde se implementará la GUI (Interfaz Gráfica de Usuario)

- Controladores: será la capa que recibirá las peticiones que provengan de las vistas, gestionando el flujo de datos entre las vistas y la base de datos

- Base de datos: En este elemento se almacenarán todos los datos de entrada y salida, con el objetivo de proveer, de dichos datos, al resto de elementos cuando sea necesario. Toda la información almacenada en este elemento se corresponderá a los dispositivos hardware alojados y emulados en la FPGA

- Interfaz PS-FPGA: elemento intermedio entre base de datos y FPGA que realiza, de forma transporte al resto de elementos, el procedimiento de formateo y transporte de la información entre base de datos y FPGA

Debido a que el sistema resultante es complejo para la parte de proceso (múltiples elementos independientes trabajando a la vez), y que es un sistema Web, se ha decidido instalar un sistema operativo por debajo de todas las capas de proceso descritas, y un servidor HTTP, con el objetivo de posibilitar y facilitar la implementación del proyecto.

FPGA

Es la capa lógica del sistema. Esta aloja los dispositivos diseñados y dispuesto a emular. Cada dispositivo emulado tiene un número concreto de registros o señales. Estas señales pueden ser de entrada o de salida, y es donde se almacenan los datos que el dispositivo requiere para realizar su función determinada.

Las señales de entrada son los parámetros con los que el dispositivo realiza el trabajo, y las de salida, son donde se retorna el resultado tras completarse dicho trabajo.

La FPGA, con el objetivo de permitir el acceso desde la parte de proceso, mapea los dispositivos emulados en memoria física. Por lo que para acceder a dichas señales hay que realizar accesos a memoria.

3.3 Tecnología hardware

Para poder llevar a cabo la implementación del proyecto, se necesita elegir una serie de arquitecturas hardware que cubran las necesidades de este. Por una parte, se necesita un dispositivo que permita procesar los elementos diseñados en el apartado 3.2 (Vistas, Controladores, Base de datos y la lógica), y por otra parte se necesita una arquitectura FPGA para emular los dispositivos.

Finalmente se ha decidido implementar el proyecto bajo una arquitectura FPGA basada en SoC. Esto se debe a que permite tener en un mismo dispositivo tanto la parte de proceso como la de FPGA.

La tabla 3-1 recoge el análisis de las características de las tecnologías hardware candidatas a ser elegidas para el proyecto. Éstas se encuentran comparadas por las características que requiere este proyecto (procesador, memoria RAM, FPGA, Ethernet y precio).

Dispositivo	Zybo Zynq-7000[13]	ZedBoard[9]	ZC702[14]
Procesador	650 MHz dual-core Cortex™-A9 processor		
Memoria RAM	512 MB DDR3		1GB DDR3
FPGA	Xilinx Zynq-7000 (XC7Z010-1CLG400C)	Xilinx Zynq-7000 (XC7Z020-CLG484-1)	Xilinx Zynq-7000
Ethernet	Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY		
Precio	168€	440€	795€
SD	Sí		

Tabla 3-1 Tecnologías hardware

Analizando la tabla 3-1, se ha decidido elegir la arquitectura hardware Zybo Zynq-7000 como tecnología hardware para implementar el proyecto. Esto se debe a que responde a unas características muy similares a la de los demás por un precio mucho más reducido (concretamente 2 ,6 veces inferior con respecto a ZedBoard, y 4,7 respecto a ZC702), cosa que se amolda perfectamente para un proyecto de investigación, aprendizaje y desarrollo como este.

Las otras dos tecnologías son mucho más caras porque responden a unas características superiores en otros aspectos no analizados, como por ejemplo el número de periféricos incorporados, potencia de la tarjeta gráfica o de sonido, etc.

En cualquier caso, debido a que el proyecto se ha diseñado de forma genérica, este puede ser implementado en cualquiera de los anteriores dispositivos mencionados en la tabla 3-1, por lo que permite cambiar de arquitectura hardware en caso de querer ampliar capacidades.

3.4 Tecnología software

Para poder llevar a cabo la implementación del proyecto, debemos identificar y elegir el tipo de tecnología software más adecuado para el desarrollo de cada uno de sus elementos. Por lo tanto, en este apartado se va a definir que tecnología software elegir como sistema operativo, como servidor, tipo de base de datos, y los lenguajes de programación para implementar las diferentes capas del sistema.

3.4.1 Sistema operativo

Debido a que el proyecto a implementar es un sistema complejo para la lógica de proceso, se va a instalar un sistema operativo por debajo del sistema implementado.

Se ha decidido restringir el sistema operativo a distribuciones basadas en Linux. Esto se debe a que son más eficientes para alojar el servidor ya que hay distribuciones específicas para sistemas empotrados. Además por las aplicaciones y librerías que Linux aporta para implementaciones como la de este proyecto (librerías de acceso a memoria física, programación de FPGA, etc). Los Linux candidatos a ser elegidos son Xillinux y Archlinux.

Xillinux

Xillinux[15] es una distribución basada en Ubuntu y adaptada específicamente por Xillybus, para la integracion sobre arquitecturas hardware FPGA basadas en SoC , específicamente sobre Zybo y Zedboard.

Esta distribución da soporte completo para la implementación de diseños hardware sobre la FPGA, dando al usuario una herramienta específica para el diseño e integración de dispositivos sobre la plataforma.

Al ser un Linux basado en Ubuntu, dispone de todas sus características. Positivas, como interfaz gráfica de usuario y el posible acceso a sus aplicaciones. Negativas para un sistema empotrado, como que es una distribución bastante pesada y que requiere muchos recursos del sistema. Está más dedicada a ser una máquina de desarrollo que de servidor. Tiene una costosa licencia de uso.

Archlinux

Archlinux[16] es una distribución Linux de código abierto. Es una distribución muy ligera (trae las aplicaciones básicas de Linux), muy personalizable (instalación específica) y, como consecuencia, consume muy pocos recursos en ejecución, por lo tanto muy recomendada para sistemas pequeños como los sistemas empotrados.

Sólo permite comandos por consola. Los comandos son considerablemente diferentes a los de las demás distribuciones. Para poder trabajar con sistemas FPGA basados en SoC, se debe combinar con el kernel específico que el fabricante pone a disposición para este tipo de dispositivos.

Análisis

Finalmente se ha decidido elegir Archilinux como sistema operativo. Esto se debe a que es una distribución con licencia de uso gratuita y con pocos requerimientos de recursos, cosa necesaria para este proyecto, en el cual se dispone de una máquina de trabajo muy limitada.

3.4.2 Servidor HTTP

El servidor que se ha decidido instalar es el servidor Apache[17], ya que da soporte a las características necesarias por el proyecto: recibir peticiones HTTP, proveer al usuario de las vistas y gestionar la capa de controladores en PHP.

3.4.3 Base de datos

Debido a que se requiere guardar la información de los dispositivos alojados en la FPGA, se ha decidido instalar una base de datos.

La base de datos elegida es Mysql[18] , los motivos son que es una base de datos muy estandarizada y es de uso libre.

3.4.4 Lenguajes de programación

Vistas

Para programar las vistas se ha utilizado HTML + Javascript. El motivo principal es que permite realizar, de forma fácil, vistas muy accesibles para el usuario. Además, permite hacer peticiones asíncronas y síncronas de forma muy sencilla.

Controladores

Para implementar los controladores se ha utilizado PHP. El motivo principal es que permite gestionar de forma fácil, tanto las peticiones provenientes de las vistas como las consultas a la base de datos.

Base de datos

Para programar tanto las tablas de la base de datos como las consultas a ésta, se ha utilizado el lenguaje SQL.

Interfaz PS-FPGA

Para implementar este elemento del sistema se ha utilizado C. El motivo principal es que es un lenguaje potente que permite realizar la lógica necesaria para poder resolver tanto las peticiones a la base de datos como la escritura en los elementos de la FPGA. Se ha utilizado principalmente para escribir en los registros de los dispositivos de la FPGA.

FPGA (capa lógica)

Para la implementación de los dispositivos o diseños hardware de la FPGA se ha utilizado el lenguaje descriptivo VHDL.

3.4.5 Entorno de desarrollo para FPGA

Como entorno de desarrollo de los elementos de la FPGA se ha utiliza Vivado. Es un software desarrollado por Xilinx para la síntesis y análisis de diseños hardware[22]. Para utilizarla se requiere una configuración específica para la FPGA con la que se va a trabajar (esta configuración la provee el fabricante). Ya configurado el entorno, esta herramienta nos permite de forma visual, a través de su interfaz gráfica de usuario, realizar diferentes funciones como: añadir nuevos

dispositivos, quitarlos, configurar FPGA o crear dispositivos propios a través de VHDL.

3.5 Diseño y desarrollo de la funcionalidad

En este apartado se va exponer el diseño y desarrollo de las funcionalidades especificas del sistema de las que dispondrá el usuario: subida de fichero de configuración FPGA, carga de fichero de configuración, registro de dispositivos en la base de datos, borrado de dispositivos de la base de datos y control de los dispositivos.

3.5.1 Diseño y desarrollo funcionalidad de subida de fichero de configuración.

Para que en la FPGA se puedan emular dispositivos se debe poder reconfigurar la FPGA siempre que sea necesario, por lo que se ha creado una funcionalidad para poder subir ficheros de configuración al sistema. La figura 3-2 muestra el diseño del flujo de esta funcionalidad.

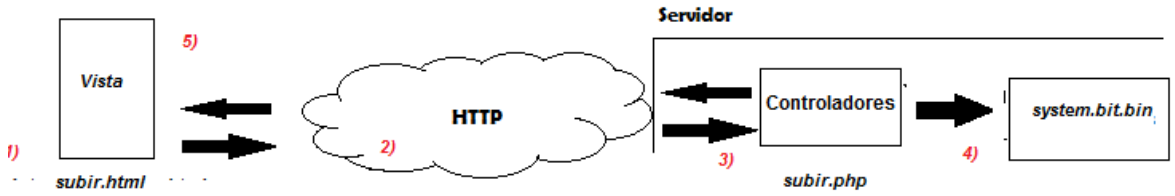


Figura 3-2 Diseño funcionalidad subida de fichero de configuración. 1) Formulario de subida de fichero. 2) Envío de formulario al servidor. 3) Controlador de subida de fichero comprueba que el fichero es correcto. 4) Si es correcto se guarda en el sistema. 5) Mensaje de respuesta.

Desarrollo de la vista

La interfaz de usuario de subida del fichero de configuración debe proporcionar al usuario una forma fácil e intuitiva de hacer dicha acción. Para ello debe tener un formulario que contenga un elemento que permita seleccionar el archivo a subir y un botón de envío que accione la subida del fichero. La figura 3-3 muestra la página creada.

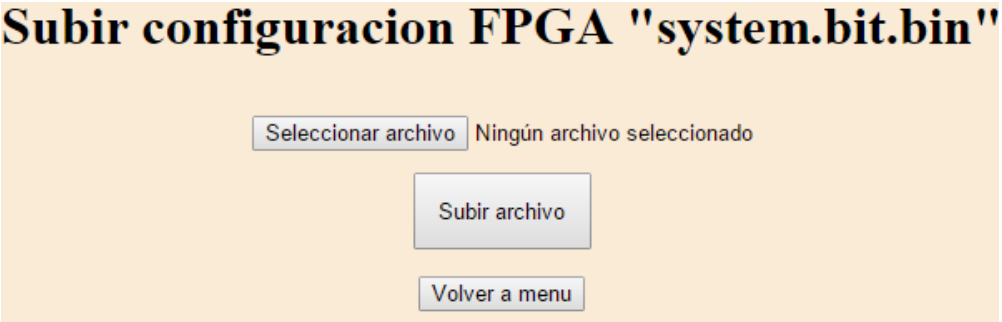


Figura 3-3 Vista de subida de archivo de configuración

Desarrollo del controlador

Como restricciones en la subida del fichero se establece que no pueda ocupar más de 10 MB y que debe llamarse "system.bit.bin". EL flujo de funcionamiento de este elemento se presenta en le figura 3-4

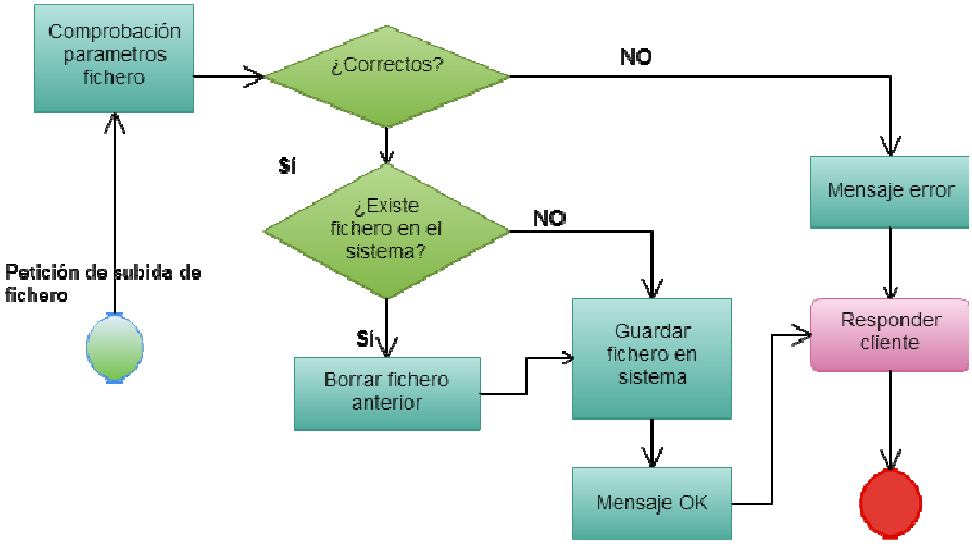


Figura 3-4 Diagrama de flujo de controlador de la subida del archivo.

3.5.2 Diseño y desarrollo funcionalidad de cargado de fichero de configuración en la FPGA.

Para que en la FPGA se puedan emular dispositivos se debe poder reconfigurar siempre que sea necesario, por lo que se ha creado una funcionalidad para permitir al usuario cargar ficheros de configuración al en la FPGA. La figura 3-3 muestra el diseño del flujo de esta funcionalidad.

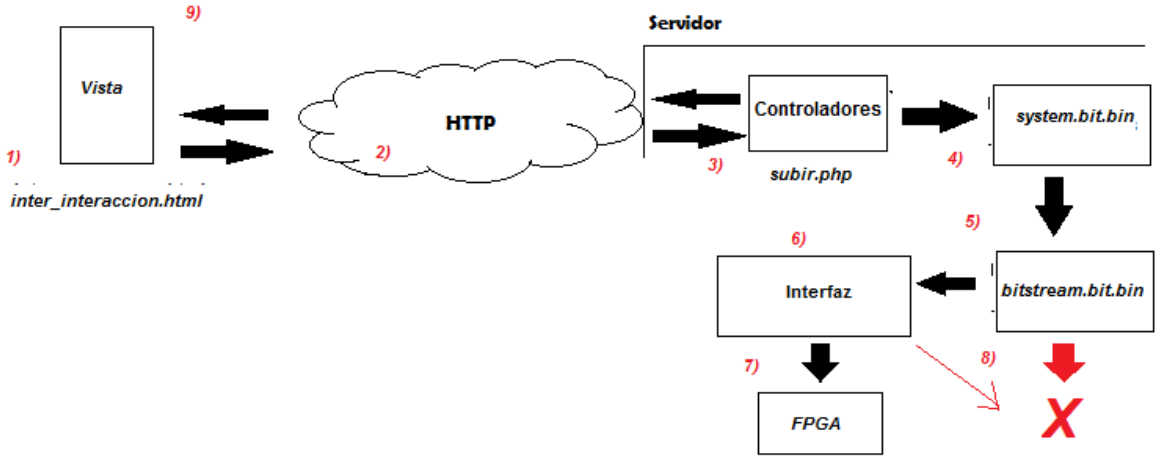


Figura 3-3 Diseño funcionalidad cargar fichero de configuración. 1) Formulario para ordenar cargado de fichero. 2) Envío de formulario al servidor. 3) Controlador de cargar fichero comienza su proceso. 4) Controlador busca el fichero de configuración. 5) Controlador copia el fichero de configuración a uno temporal. 6) El elemento de la lógica encuentra el fichero temporal. El

elemento de lógica carga el fichero en la FPGA. 7) El elemento de lógica borra el fichero temporal.

Desarrollo de la vista

El desarrollo de esta vista está integrado en la vista de "Control de dispositivos".

Desarrollo del controlador

Copia el fichero de configuración anteriormente subido creando uno temporal y que se llamará de una forma determinada la cual el demonio PS-FPGA espera encontrar. El flujo de funcionamiento de este elemento se presenta en el la figura 3-5

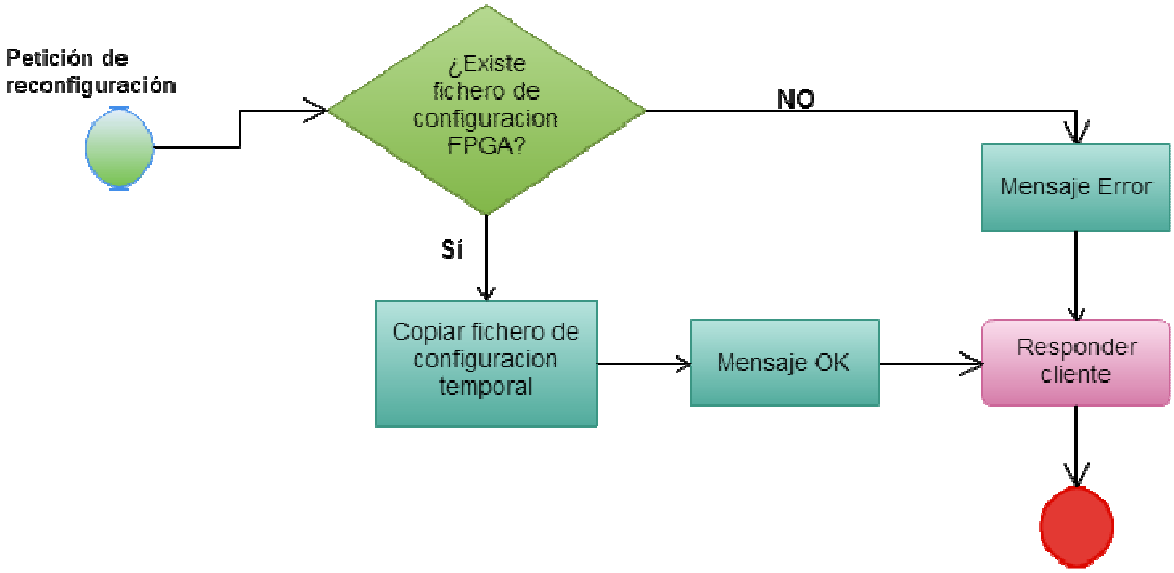


Figura 3-5 Diagrama de flujo reconfiguración FPGA

3.5.3 Diseño y desarrollo funcionalidad de registro de dispositivos

Para que el sistema reconozca los dispositivos alojados en la FPGA, y pueda almacenarse la información de estos, es necesario crear una funcionalidad que permita el registro de los dispositivos en la base de datos. La figura 3-6 muestra el diseño del flujo de esta funcionalidad.

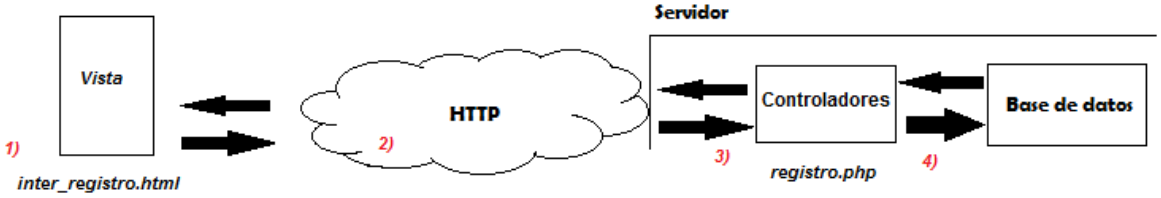


Figura 3-6 Diseño funcionalidad registro dispositivos. 1) Formulario de registro. 2) Envío de formulario al servidor. 3) Controlador de registro

comprueba los datos recibidos y registra, si se puede, el dispositivo en la base de datos. 4) Mensaje de respuesta.

Desarrollo de la vista

La página de registro, a modo de formulario, permite de forma intuitiva rellenar los campos para realizar un correcto registro de un dispositivo concreto en el sistema.

Consta de dos campos concretos para la parte del registro del dispositivo: nombre de dispositivo y dirección del dispositivo en el sistema.

Para la parte de registro de señales, como éstas son un número indeterminado para cada dispositivo, entonces se ha decidido diseñar un campo de selección de número de señales, con el objetivo de generar de forma dinámica los campos necesarios para cada señal. Cada señal tiene dos campos concretos: el tipo de señal y si ésta es de entrada o de salida.

Como puede observarse en la figura 3-7, esta vista tiene una barra de desplazamiento para permitir ver el formulario si este crece demasiado.

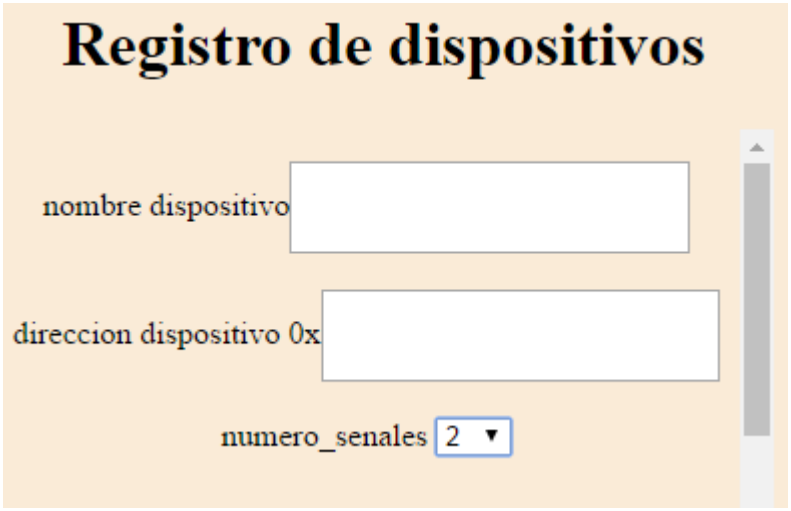


Figura 3-7 Página de registro de dispositivo

Desarrollo del controlador de registro

El desarrollo de este controlador sigue el diagrama de flujo de la figura 3-8.

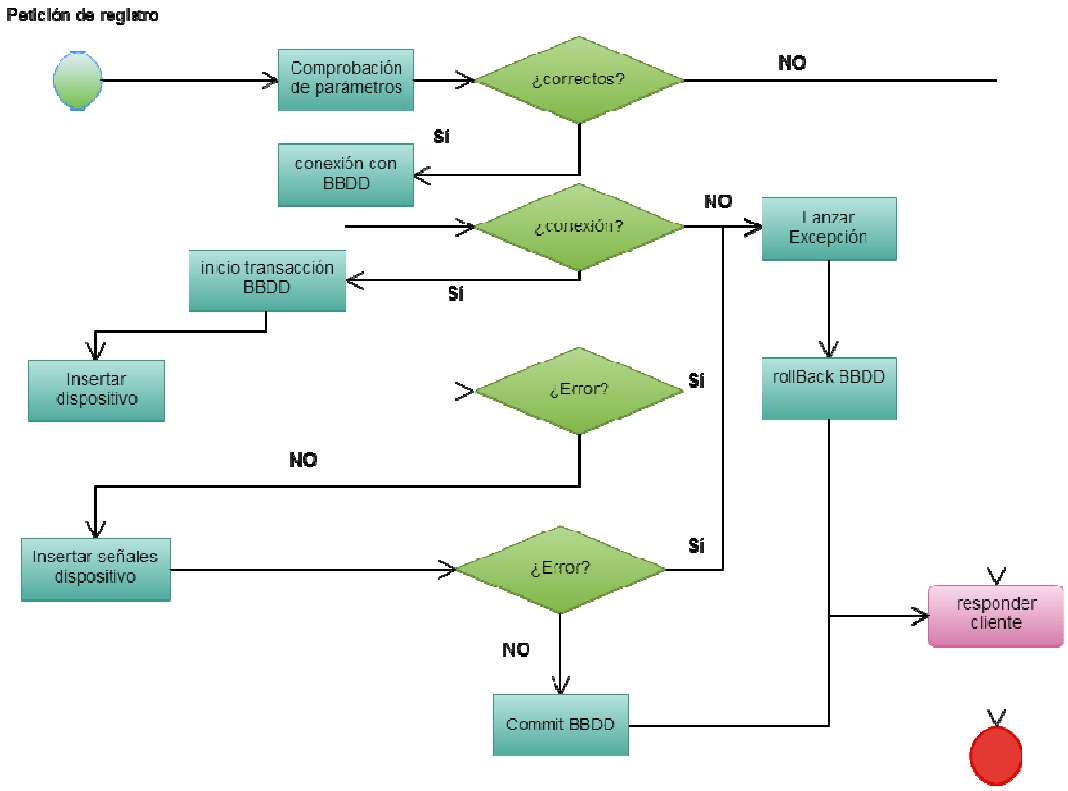


Figura 3-8 Diagrama de flujo – controlador de registro

3.5.4 Diseño y desarrollo funcionalidad de control de dispositivos

Esta funcionalidad es uno de los objetivos más importantes del TFG. Ésta implementa un flujo de ejecución que permite controlar los diseños hardware emulados en la FPGA, de tal forma que se puedan parametrizar y obtener sus resultados. La figura 3-5 ilustra el diseño de esta ejecución.

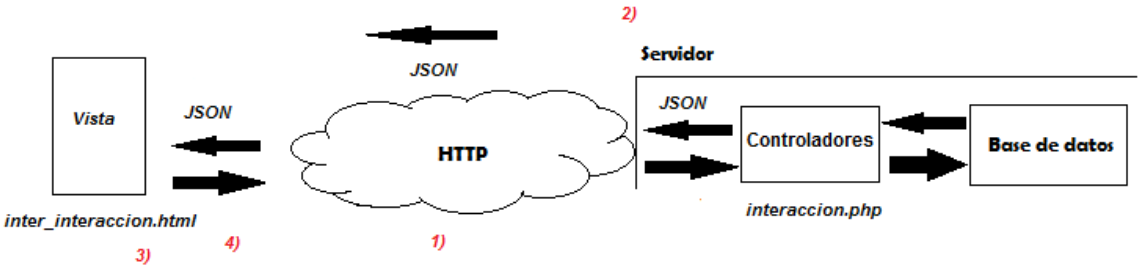


Figura 3-5 Diseño funcionalidad control dispositivos. 1) Petición asíncrona para pedir los dispositivos disponibles (ejecutada automáticamente cada x segundos). 2) El controlador responde con un JSON[20] (JavaScript Object Notation) de respuesta que contiene todos los dispositivos con su información. 3) Se genera el formulario de los dispositivos. 4) Se envían parametrizaciones de los dispositivos.

Desarrollo de la vista de control de dispositivos

Esta página, se ha diseño con intención de permitir leer de forma automática los dispositivos, y con la posibilidad de actualizar sus señales de forma intuitiva y fácil, a través de una interfaz gráfica. Para esto se necesita un proceso asíncrono, que de forma dinámica y automática genere la interfaz gráfica con los dispositivos disponibles y, que de forma periódica, se vuelva a actualizar de la misma forma.

La estructura de la página se ha diseñado en tres bloques heterogéneos, formando así tres columnas bien definidas. De izquierda a derecha en la figura 3-9.

- En el primer bloque se introducido un selector de segundos que permite al usuario seleccionar el tiempo entre actualización (20 – 300 segundos) y también un botón para permitir realizar la orden de carga de fichero de configuración en la FPGA.

- El segundo bloque da la posibilidad al usuario de seleccionar un dispositivo concreto y actualizar las señales de éste en el sistema. Este bloque, se genera de forma dinámica y automáticamente en cada actualización.

- El tercer bloque se genera también en cada actualización. Es equivalente al segundo bloque pero con la diferencia de que es sólo de lectura de todos los dispositivos.

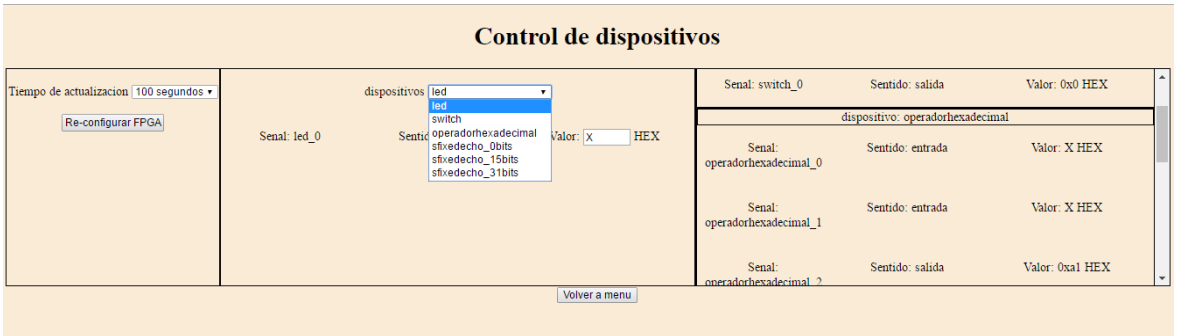


Figura 3-9 Vista – control de dispositivos

Desarrollo del controlador de control de dispositivos

El módulo que se encarga de gestionar y responder las peticiones de control de dispositivos se ha diseñado de la siguiente forma: si se reciben parámetros coherentes y correctos que indican deseo de actualizar señales de alguno o varios dispositivos, entonces el módulo procederá a intentar escribir en dichas señales de entrada en la base de datos. En cualquier caso, después de gestionar cualquier tipo de petición a este módulo, si no ha habido error, se procederá a devolver al cliente la información del estado de todos los dispositivos registrados en el

sistema. Esta información es transportada dentro es un formato ligero para el intercambio de datos, el formato JSON. Para esto tiene que recoger los datos y construir la estructura de datos JSON. La figura 3-10 muestra el flujo de ejecución de este diseño.

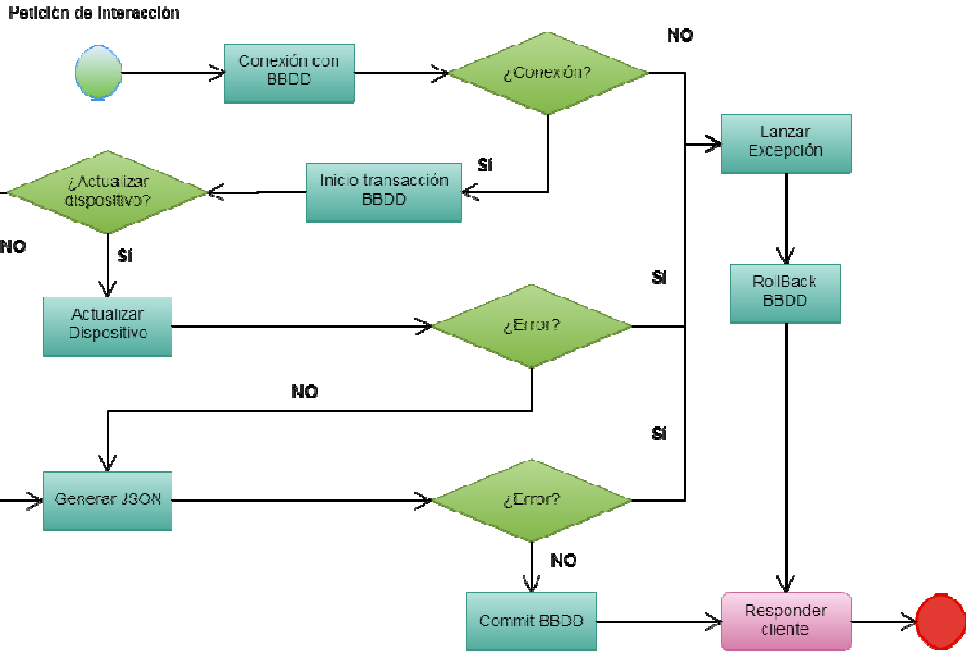


Figura 3-10 Diagrama de flujo – controlador de control de dispositivos

3.5.5 Diseño y desarrollo funcionalidad de borrado de dispositivos

Para que el sistema deje de reconocer los dispositivos alojados en la FPGA, es necesario crear una funcionalidad que permita el borrado de los dispositivos de la FPGA. La figura 3-11 muestra el diseño del flujo de esta funcionalidad.

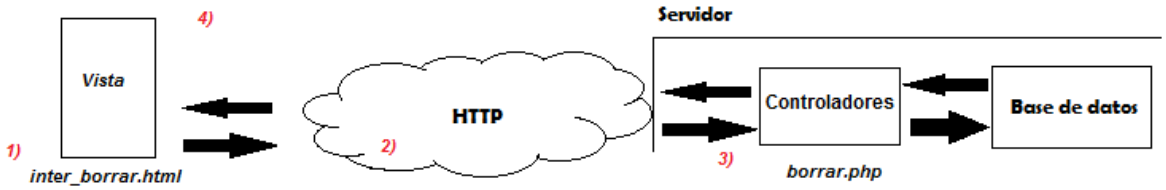


Figura 3-11 Diseño funcionalidad borrado dispositivos. 1) Formulario de borrado.2) Envío de formulario al servidor. 3) Controlador de registro comprueba los datos recibidos y borrar, si se puede, de la base de datos. 4) Mensaje de respuesta.

Desarrollo de la vista

La interfaz gráfica de usuario debe tener los campos necesarios para dar la posibilidad a este de poder borrar, o un dispositivo concreto, o todos los dispositivos a la vez. Entonces como se puede intuir, esta página constará de esas dos posibilidades, en el primer caso: un campo de texto para el nombre de dispositivo y un botón asociado a ese campo. La figura 3-12 muestra el resultado final del diseño.

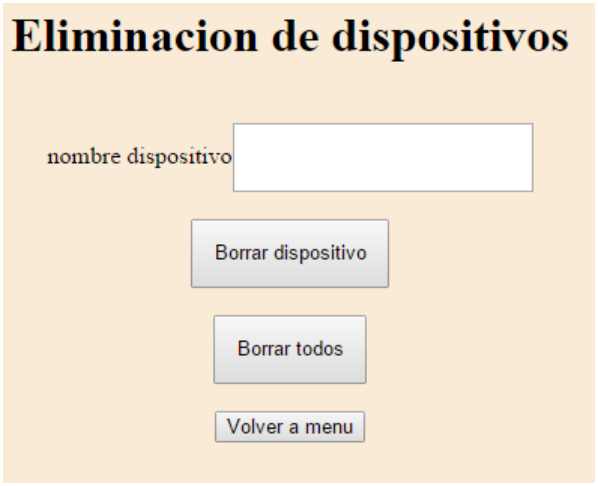


Figura 3-12 Vista – Eliminación de dispositivos

Desarrollo del controlador

La figura 3-13 muestra el flujo de ejecución del controlador de borrado de dispositivos.

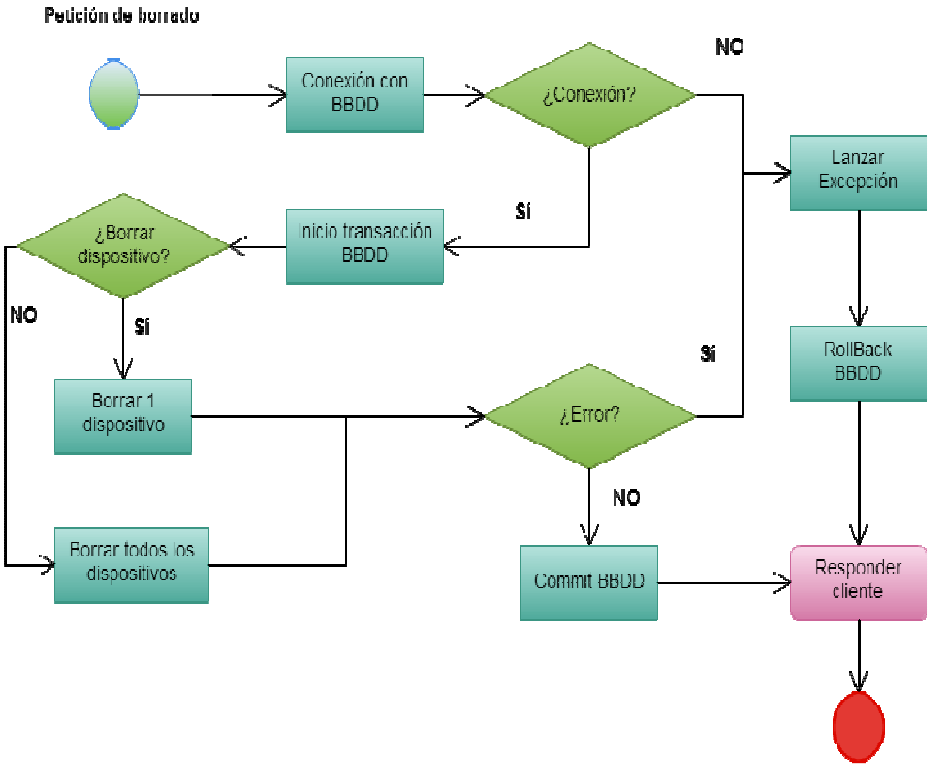


Figura 3-13 Diagrama de flujo – controlador borrar dispositivos

3.5.6 Diseño y desarrollo funcionalidad de menú de navegación

Para que el sistema deje de reconocer los dispositivos alojados en la FPGA, es necesario crear una funcionalidad que permita el borrado de los dispositivos de la FPGA. La figura 3-4 muestra el diseño del flujo de esta funcionalidad.

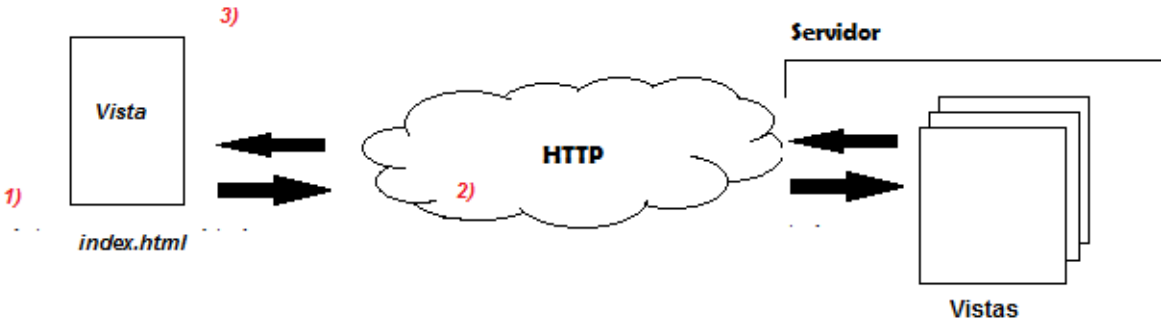


Figura 3-14 Diseño funcionalidad menú de navegación. 1) Menú de navegación a las páginas. 2) Envío de petición. 3) Cambio de vista

Desarrollo de la vista.

En la figura 3-15 se muestra el resultado de esta página, a modo de menú, permite la navegación entre páginas para la interacción completa con el sistema.

Figura 3-15 vista – menú de navegación

3.6 Diseño de la base de datos

La base de datos sigue el diagrama entidad-relación de la figura 3-16. La entidad dispositivo representa la información relevante de los dispositivos alojados en la FPGA. La entidad señal representa la información de las diferentes señales ligadas a un dispositivo.

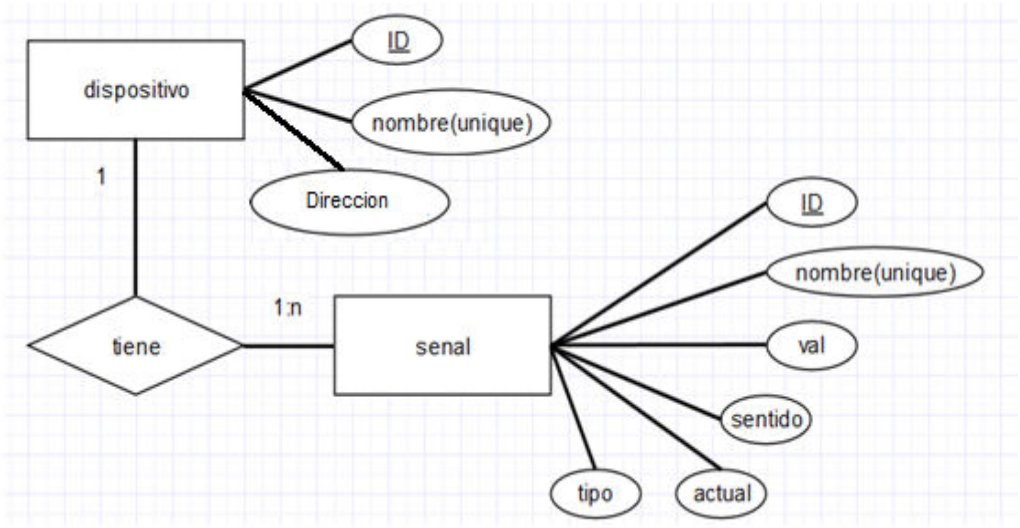


Figura 3-16 Diagrama entidad relación

3.7 Interfaz PS-FPGA

En esta sección se va a explicar el diseño y desarrollo del elemento responsable de la comunicación entre Base de datos y FPGA.

3.7.1 Diseño

A través de este elemento se debe permitir realizar dos cosas: intercambiar información entre la base de datos y la FPGA, y realizar, cuando sea necesario, el proceso de cargado de un fichero de configuración en la FPGA. A parte de funcionalidad este elemento debe proporcionar disponibilidad continua, por lo que debe desarrollarse como un proceso demonio. La figura 3-17 indica la corriente de datos que realiza el proceso de la interfaz.

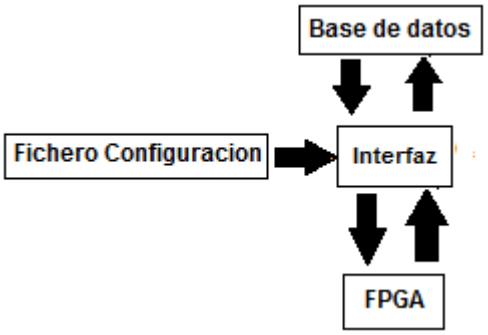


Figura 3-17 Flujo de datos capa lógica

3.7.2 Desarrollo

Siguiendo las indicaciones del diseño se ha generado dicho elemento. La Figura 3-18 indica el diagrama de flujo principal del algoritmo programado. Debido a su complejidad, se han eliminado algunos flujos no importantes para esclarecer la presentación.

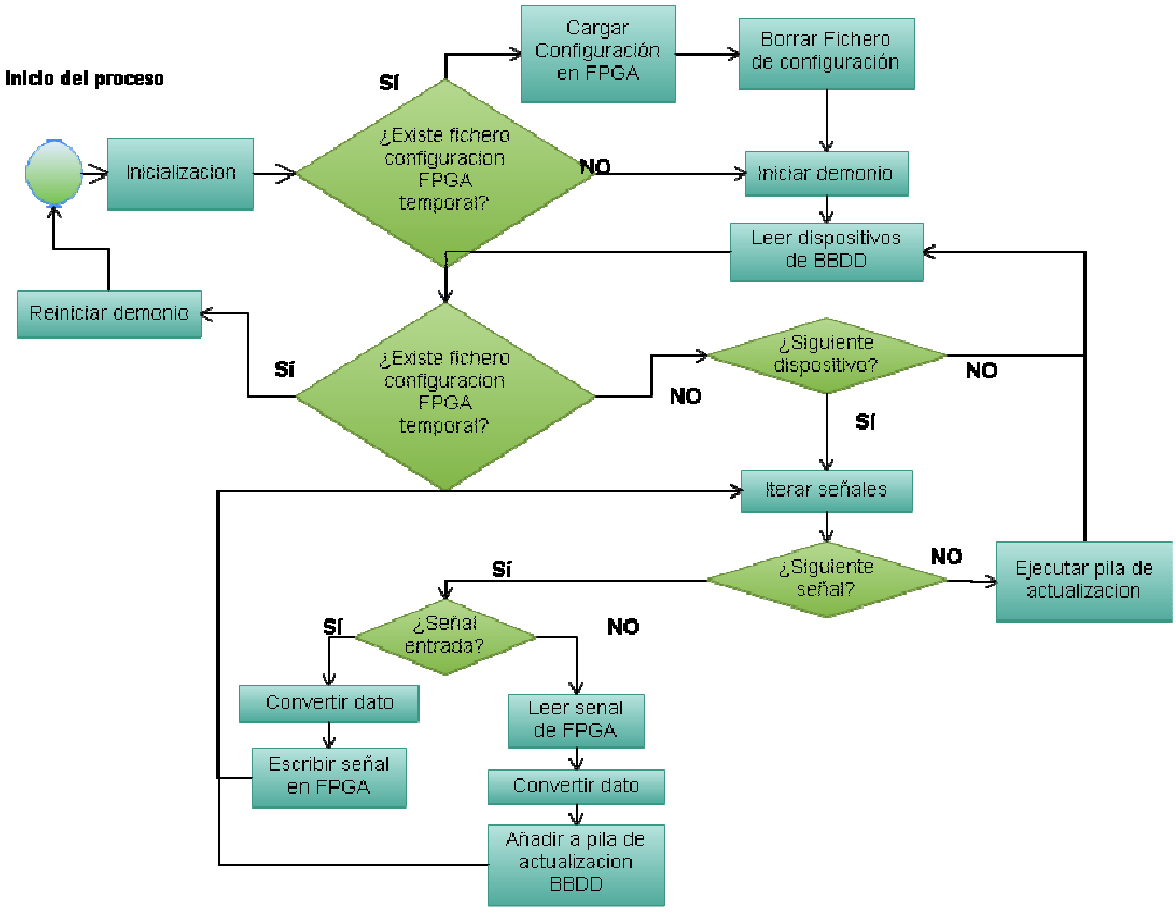


Figura 3-18 Diagrama de flujo Interfaz PS-FPGA

Este elemento para llevar a cabo todas sus funciones está formado por cinco módulos:

Módulo demonio
Este módulo se encarga de convertir el proceso en un demonio. Esto se realiza mediante la bifurcación de este, en proceso padre y proceso hijo, dejando sólo el hijo en funcionamiento.
Módulo logueo
Módulo que almacena en un fichero toda la información que ocurre en el demonio.
Módulo de acceso a Base de Datos
Este módulo encapsula las llamadas a las principales funciones de acceso a la api de Mysql para C [19].

Con el objetivo de realizar múltiples consultas de actualización a la vez, se ha implementado una pila de almacenamiento de éstas. Para interactuar con ella existen dos funciones: añadir consultas a la pila y ejecutar todas las consultas existentes en ésta.

Módulo de conversión
Desde la base de datos se pueden recibir datos en diferentes tipos de representación, pero la FPGA sólo trabaja con números en representación binaria.

Esto quiere decir que previamente antes de una comunicación entre base de datos y FPGA es necesaria una conversión de tipos.

Este módulo provee de funciones que permiten realizar conversiones entre diferentes tipos de representación de datos. La tabla 3-2 muestra los tipos de conversión posibles.

Representación origen	Representación destino	Argumentos
Hexadecimal	Binario	-
Binario	Hexadecimal	-
Decimal	Coma Fija con signo	Numero de bits dedicados a la parte decimal (31 máx)
Coma Fija con signo	Decimal	Numero de bits dedicados a la parte decimal (31 máx)

Tabla 3-2 Tabla de posibles conversiones

Módulo de comunicación con FPGA

Este módulo se ha desarrollado a modo de API(Application Programming Interface). A través de una serie de funciones, permite escribir o leer registros en una determinada ubicación de memoria. El módulo hace el cálculo de la dirección efectiva a través del número de registro al cual se quiere acceder y la dirección de memoria base. El acceso a memoria se realiza a través de la función mmap[21]. Para conocer más a fondo la API consultar con el Anexo "Manual del programador".

Módulo principal

Este módulo realiza la semántica final del algoritmo, de tal forma que realiza la integración ordenada en las llamas a los módulos anteriores y lo conforma en uno sólo cuando se realiza la compilación de este elemento.

Adicionalmente mencionar, que una vez integrados todos los módulos en un mismo elemento, para ejecutar, ha sido necesario incorporar dicho elemento como un servicio en Linux[23] con el objetivo de que se ejecute en modo demonio al iniciar el sistema.

3.8 Desarrollo FPGA

En esta sección se va a explicar los diseños hardware emulados en la FPGA, y que han sido implementados para probar el sistema.

- **Dispositivo led:** este dispositivo tiene un único registro o señal. Esta es de entrada. Su función es comprobar la funcionalidad de escritura en la FPGA del módulo. Este dispositivo enciende uno de los leds de la placa al

escribir un ‘1’ en la señal de entrada. Si señal es ‘0’ entonces el led se apaga, por el contrario el led estará encendido.

- **Dispositivo switch:** este dispositivo tiene un único registro de salida. Su objetivo es comprobar la funcionalidad de lectura en la FPGA. Este dispositivo lee uno de los switch que la placa o circuito integrado tiene. Cuando dicho switch está en una posición, el valor de la señal que se leerá será ‘0’, cuando este esté en la posición contraria entonces el valor leído será ‘1’.
- **Dispositivo sumador Hexadecimal**
  El objetivo de este dispositivo es sumar dos números en hexadecimal. Para esto este dispositivo consta de tres señales. Dos de entrada, números a sumar, y una de salida, resultado de la suma. Con este dispositivo se probará la conversión a hexadecimal.
- **Dispositivo repetidor de señal**
  El objetivo de este dispositivo es hacer un eco y repetir un número real insertado a la entrada. Contiene dos señales: una de entrada (número a repetir), una de salida con el número repetido. Este dispositivo probará el conversor de coma fija, de número real en notación decimal a formato en bits de coma fija.

4 Integración, pruebas y resultados

En este capítulo se va a proceder a realizar las pruebas sobre el sistema final. Para ello se accederá explícitamente a las funciones que proporciona el sistema para el usuario.

4.1 Menú de navegación

La figura 4-1 muestra las secciones disponibles para el usuario. Estas secciones se corresponden con las funcionalidades del sistema a probar.

Figura 4-1 Vista menú

Se parte con la premisa de que en la FPGA están cuatro dispositivos cargados y dispuestos a poder emularse. La tabla 4-2 ilustra los cuatro dispositivos mencionados.

Dispositivo	Dirección	Nº señales entrada	Nº señales salida
Led	0x40000000	1 hexadecimal	0
Switch	0x43C00000	0	1

			hexadecimal
OperadorHexadecimal	0x43C30000	2 hexadecimal	1 hexadecimal
SfixedEcho	0x43C20000	1 coma fija	1 coma fija

Tabla 4-2 Dispositivos emulados en FPGA

4.2 Registro de dispositivos

Para registrar dichos dispositivos se accede a la sección de "registro de dispositivos", y se muestra la vista necesaria para realizar dicho registro en la figura 4-2. Se va a proceder a registrar todos los dispositivos, sólo se va a mostrar visualmente el registro de los dispositivos "operador hexadecimal" y "SfixedEcho".

Registro "operador hexadecimal"

La figura 4-2 muestra la vista completa al realizar el registro del dispositivo "operador hexadecimal".

Figura 4-2 Vista de Registro. Ejemplo de registro completo del dispositivo "operador hexadecimal".

Registro "SfixedEcho"

Este dispositivo puede probarse con diferentes configuraciones de bits para su conversión en el tipo de representación, por lo que se registrará varias veces y con diferentes configuraciones: 0bits, 15bits y 31bits para la parte decimal. La figura 4-3 muestra un ejemplo para realizar el registro del dispositivo "SfixedEcho".

Figura 4-3 Vista de Registro. Ejemplo de registro del dispositivo "SfixedEcho" con configuración de 0bits.

4.3 Control de los dispositivos

Se procede a probar los dispositivos registrados y latentes en la FPGA. Para ello hay que acceder a la página "control de dispositivos", fácilmente accesible desde el menú de navegación. La página resultante aparece es la mostrada en la figura 4-4.

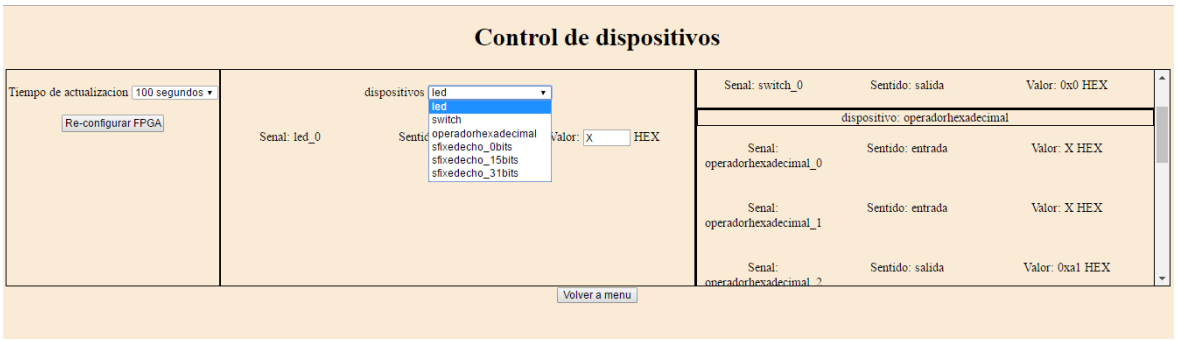


Figura 4-4 Vista de control de de dispositivos con toda la información de los dispositivos registrados.

4.3.1 Pruebas con el dispositivo "led"

Este dispositivo sabemos que enciende y apaga un led. Partimos con el led apagado.

En la página de control introducimos un '1' para intentar encender el led y pulsamos actualizar, la Figura 4-5 comprueba que se ha encendido el led.

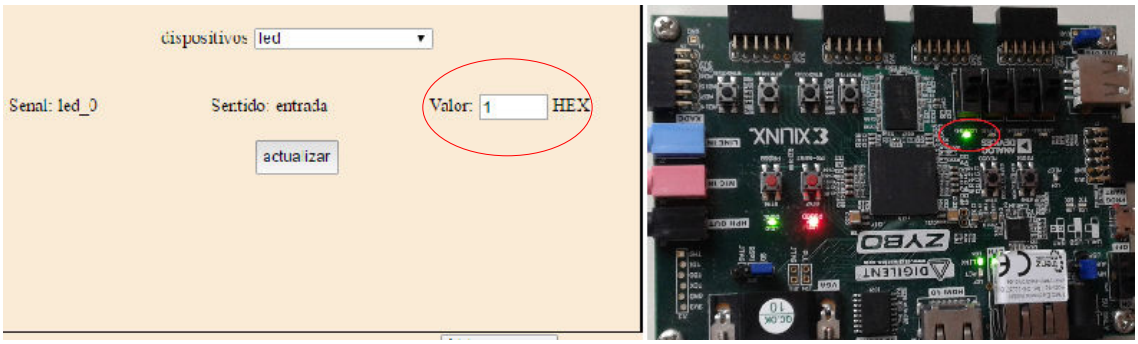


Figura 4-5 Vista de control de dispositivos – pruebas "led"

4.3.2 Pruebas con el dispositivo "switch"

Movemos de posición el switch de la tarjeta. En la figura 4-6 se puede comprobar que el valor de la señal del dispositivo "switch" ha cambiado a '0x1'.

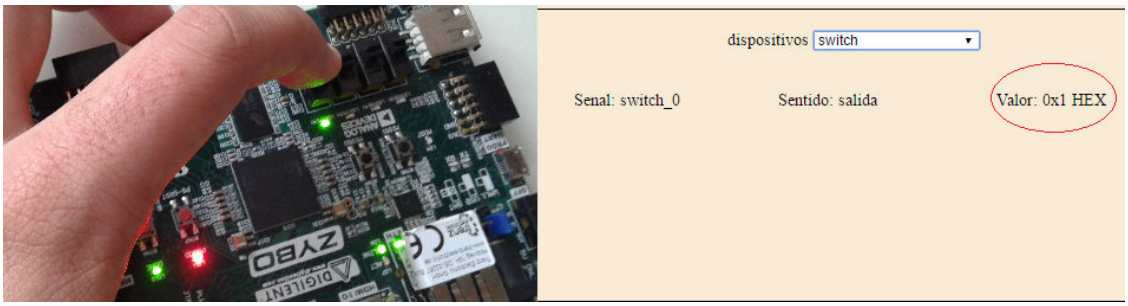


Figura 4-6 Vista de control de dispositivos – pruebas "switch"

4.3.3 Pruebas con el dispositivo "OperadorHexadecimal"

En este caso la función alojada en esta posición de memoria es un sumador de dos números en hexadecimal. La tabla 4-1 ilustra las pruebas que se van a realizar.

Parámetros de entrada	Salida
0x5, 0x1A	0x1F
0xFFF, 0x1	0x1000
0xA, 0xA	0x14

Tabla 4-1 Pruebas "operador hexadecimal"

La figura 4-7 muestra algunos ejemplos de las pruebas propuestas en la tabla 4-1.

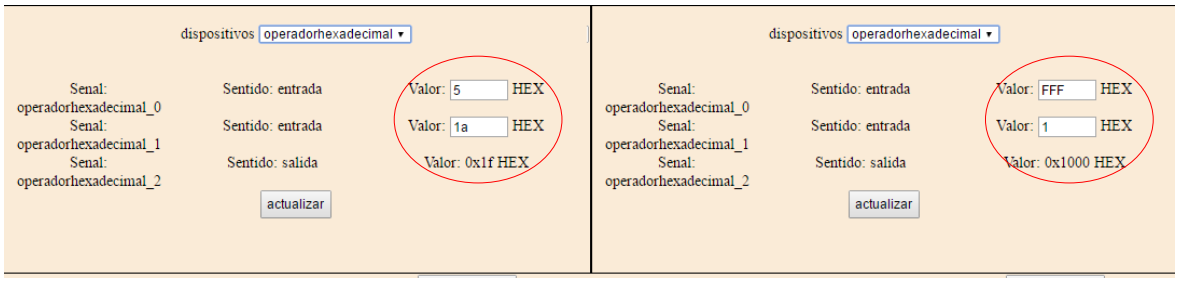


Figura 4-7 Vista control dispositivos – pruebas "operador hexadecimal"

4.3.4 Pruebas con el dispositivo "SFixedEcho"

Con este dispositivo tenemos tres configuraciones posibles a probar: 0bits, 15bits y 31bits. Además tenemos que probar con números negativos y positivos. La tabla 4-2 muestras las pruebas propuestas para este dispositivo.

Configuración	Entrada	Respuesta Esperada ~
SFixedEcho_0bits	99,55	99,00
	-99,55	-99,00
	0,555	0,0
	500,0	500,0
SFixedEcho_15bits	99,55	99,55
	0,555	0,555
	500,0	500,0
SFixedEcho_31bits	99,55	0,55
	0,555	0,555
	500,0	0,0

Tabla 4-2 Pruebas SFixed

La figura 4-8 muestra algunos ejemplos que reproducen la tabla 4-2.

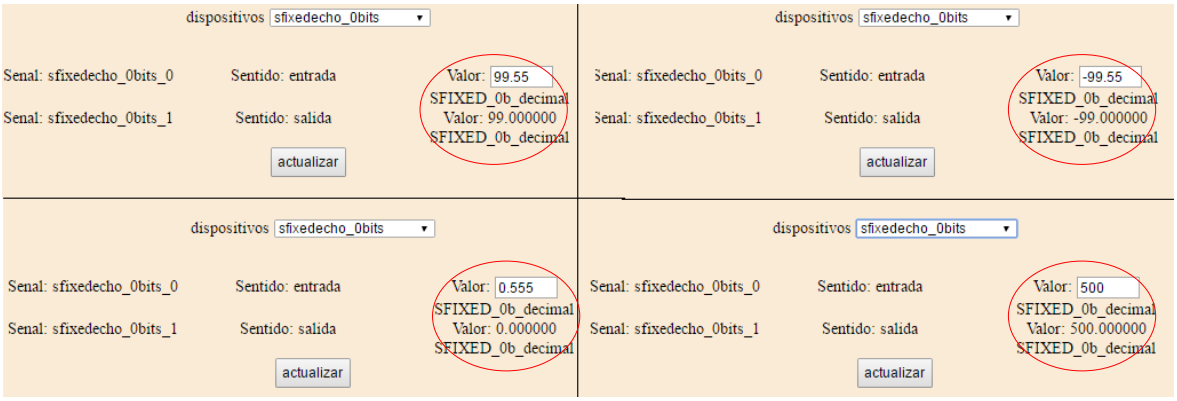


Figura 4-8 Vista control dispositivos – pruebas "sfixedecho_0bits"

4.4 Prueba para demostrar el funcionamiento de reconfiguración de la FPGA mediante el servidor.

Se procede a probar la funcionalidad de subir fichero de configuración y reconfiguración de la FPGA con el mismo. Este archivo de configuración con el que se va a realizar la prueba tiene el mismo número de dispositivos alojados en las mismas posiciones de memoria, pero con la diferencia de que algunos de ellos realizan una función diferente a la que hacían con la configuración previa. Los dispositivos que realizan una función diferente son el "Led" y el "OperadorHexadecimal".

Led: con la configuración anterior el led se encendía con una operación lógica directa, encendía en '1' y se apagaba en '0'. Con la nueva, la operación lógica será negada, es decir, se enciende en '0' y se apaga en '1'.

OperadorHexadecimal: con la configuración anterior se realizaba una suma aritmética entre dos números en hexadecimal. Con la nueva, la función suma se ha sustituido por la función de resta, por lo que se realizara una resta en hexadecimal.

4.4.1 Subida del archivo de configuración de FPGA.

Se procede a ir a la página del menú "Subir archivo de configuración FPGA (system.bit.bin)" y a subir el archivo "system.bit.bin" que se desea probar. El sistema indica que la subida del fichero es correcta.

4.4.2 Reconfiguración de la FPGA

Ahora se procede a ir otra vez a la página de control de los dispositivos y a pulsar el botón "reconfigurar FPGA". Este botón realiza la petición de orden de reconfigurar la FPGA al sistema web con el fichero de configuración previamente subido. Si se ha realizado con éxito dicha configuración, aparecerá un mensaje como el que ilustra la figura 4-9.

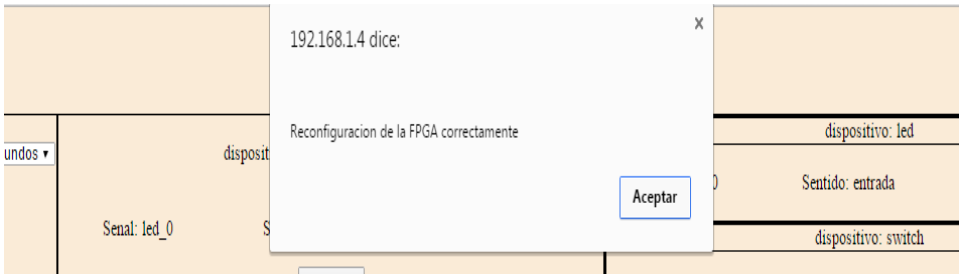


Figura 4-9 Vista de control de dispositivos – reconfigurando FPGA

4.4.3 Verificación

Ahora se supone que la FPGA se ha reconfigurado correctamente con los nuevos dispositivos del fichero subido anteriormente. Para comprobarlo se procede a probar los dispositivos que deben haber cambiado con dicha configuración.

Led

Vemos que ahora si ponemos el parámetro de entrada del valor de su señal a '0' este se enciende. Este resultado se puede comprobar en la figura 4-10:

Figura 4-10 Vista de control de dispositivos – prueba "led"

OperadorHexadecimal

Se procede a probar que este dispositivo ya no realiza la suma sino que realiza restas. La última prueba con este dispositivo era la parametrización con los números '0xA' y '0xA', y con resultado '0x14' (ver en la sección de pruebas con este dispositivo antes de la reconfiguración). Ahora el resultado deber ser '0x0', ya que ahora realiza la función resta en vez de la de suma. La prueba de este resultado se puede comprobar en la figura 4-11.

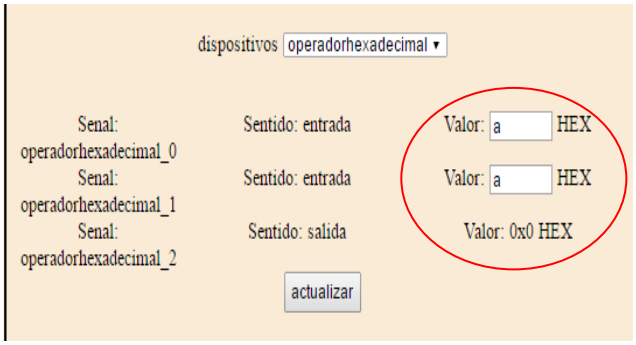


Figura 4-11 Vista de control de dispositivos - pruebas "operadorHexadecimal"

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Se buscaba implementar un marco de trabajo que permitiese emular dispositivos hardware en una FPGA de forma remota. Por lo que se ha implementado un sistema web diseñado bajo la arquitectura Modelo-Vista-Controlador. Para llevar a cabo estos objetivos, se requería la implementación de una serie de funcionalidades específicas: monitorización de los dispositivos emulados en la FPGA y una que permitiese reprogramar la FPGA de forma dinámica. Entonces, se han desarrollado cinco funcionalidades que dan soporte a estos requisitos: subida de un archivo de configuración al sistema, cargado de este fichero de configuración en la FPGA, registros de dispositivos para comunicar al sistema la aparición de nuevos dispositivos en la FPGA, borrado de dispositivos del sistema y monitorización de los dispositivos.

Con el objetivo de facilitar la interacción de estas funcionalidades con el usuario, era necesario proporcionar una interfaz gráfica. Para llevar a cabo estos requisitos se han desarrollado cinco páginas web que integran todas las funcionalidades descritas: página de registro, página de borrado, página de control de dispositivos y cargado de fichero de configuración, y página de subida de archivo de configuración.

Finalmente, el sistema obtenido permite los objetivos deseados, reprogramar la FPGA a través de un fichero de configuración, monitorizar los dispositivos emulados en esta, y todo ello de forma remota, por lo que puede concluirse con que se ha desarrollado un sistema completo de emulación de dispositivos de forma remota.

5.2 Trabajo futuro

- **Tipos de representación de datos para convertir dinámicos**.

Actualmente, estos tipos trabajan de forma automática en todos los elementos del proyecto pero estos están definidos de forma estática en cada uno de los elementos.

Sería interesante dotar al sistema de una definición dinámica de los tipos de representaciones de datos posibles mediante la BBDD, de tal forma que se pudiese redefinir nuevos tipos de forma dinámica y personal.

- **Aumentar funcionalidad al sistema.**

Extender la funcionalidad del sistema para que pueda realizarse el diseño de los dispositivos en línea y directamente en la plataforma del sistema.

- **HTTPS**

Dotar al sistema de protocolos de canal seguro como HTTPS.

Referencias

[1] "FPGA (Field Programmable Gate Array)" Wikipedia https://es.wikipedia.org/wiki/Field_Programmable_Gate_Array#FPGA_vs_ASICs

[2] La web https://es.wikipedia.org/wiki/World_Wide_Web

[3] Laboratorio virtual para la programación de FPGAs http://www.uco.es/~el2pamuj/docs/research/internat_cong/SIIE2005_1.pdf

[4] Quartus II https://es.wikipedia.org/wiki/Quartus_II

[5] JTAG Server http://quartushelp.altera.com/14.0/mergedProjects/program/pgm/pgm_pro_add_server.htm

[6] System On Chip https://es.wikipedia.org/wiki/System_on_a_chip.

[7] Java Servlet Page https://es.wikipedia.org/wiki/JavaServer_Pages

[8] Interfaz Gráfica de Usuario https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario

[9] Remote FPGA –SoC https://www.researchgate.net/publication/269297262_Remote_SoCFPGA_platform_configuration_for_cloud_applications

[10] ZedBoard http://zedboard.org/product/zedboard

[11] GlassFish https://es.wikipedia.org/wiki/GlassFish

[12] Microzed http://zedboard.org/product/microzed

[13] Modelo-Vista-Controlador https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[14] Zybo http://www.xilinx.com/products/boards-and-kits/1-4azfte.html

[15] ZC702 http://www.ti.com/lit/ug/slyu019a/slyu019a.pdf

[16] Xillinux http://xillybus.com/xillinux

[17] Arch Linux http://www.aurumlinux.com/distros/deberias-usar-arch-linux-si/

[18] Apache https://httpd.apache.org/

[19] Maria DB https://mariadb.org/

[20] API de Mysql para C https://dev.mysql.com/doc/refman/5.7/en/c-api-example-programs.html

[21] JSON https://es.wikipedia.org/wiki/JSON

[22] FPGA access http://fpga.org/2013/05/28/how-to-design-and-access-a-memory-mapped-device-part-two/.
[23] XilinxVivado https://en.wikipedia.org/wiki/Xilinx_Vivado
[24] Servicio Demonio Arch Linux https://wiki.archlinux.org/index.php/
[25] Loading bitfile in FPGA with xdevfcg https://forums.xilinx.com/t5/Embedded-Linux/Zynq-Loading-bitfile-into-FPGA-from-Linux-xdevcfg/td-p/237850/page/4.
[26] Linux en zybo http://blog.zembia.cl/instalando-linux-en-la-tarjeta-zybo/
[27] Digilent https://reference.digilentinc.com/zybo:zybo
[28] LAMP en Archlinux https://www.linode.com/docs/websites/lamp/lamp-server-on-arch-linux
[29] Añadir un archivo http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html

Glosario

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BBDD	Base de datos
CORS	Cross-origin resource sharing
FPGA	Field Programable Gate Array
FSBL	First Stage Boot Loader
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JQuery	Javascript Query
JSON	JavaScript Object Notation
PDO	PHP Data Objects
PL	Programable Logic
PHP	Pre Hypertext -processor
PS	Process system
SO	Sistema operativo
SoC	System on Chip
UIO	User Input Output
URL	Uniform Resource Locator
MVC	Modelo-Vista-Controlador

Anexos

A Manual de instalación

Instalación de Linux en Zybo

A priori

Este manual sirve para instalar Linux desde cero. Este sigue un tutorial externo, con comentarios adicionales, con los que se consigue una configuración para el correcto funcionamiento del sistema final. El manual ha sido probado en la tarjeta de desarrollo Zybo Zynq-7000 ARM/FPGA SoC (System on Chip). Sin embargo, y en principio, cambiando sólo un par de matices de configuración específicos, debe poder utilizarse para cualquier modelo de Zynq 7000.
Para instalar Linux en esta arquitectura, se deben generar varios ficheros específicos: el BOOT.bin(uboot+configuración Fpga+ dispositivo), uImage(kernel de Linux) y device tree.
La distribución que se ha utilizado como sistema de archivos es archlinux, ya que es una distribución básica, se instala sólo lo necesario, y como consecuencia, ocupa bastante menos que otras distribuciones existentes, esto hace que sea ideal para nuestro caso, un sistema embebido. No obstante, aunque el tutorial use archlinux, se puede utilizar cualquier otra distribución.
El tutorial a seguir está alojado en mismo directorio que este manual, o puedes acceder a él mediante este enlace tutorial [25].

Comentarios a tener en cuenta

1. Preparando la tarjeta microSD

La cantidad de memoria que se asigna a la partición ZYBO_BOOT es excesiva(1G). Con asignar 100 megas basta.

.

2. Compilar U-boot

No reservar memoria para los dispositivos. Dejarlo como está:
"fdt_high=0x20000000₩0" ₩
"initrd_high=0x20000000₩0" ₩

Aviso: la línea *make modelo_tarjeta_config*
deberá hacerse para el tipo de tarjeta en la que se va a instalar el sistema. Como en este caso utilizamos zybo, será *make zynq_zybo_config*.

3. Compilar el kernel de Linux

Cuando se ejecuta make menuconfig:

- **Soporte para adaptadores USB-WiFi**
  En principio no se va a utilizar, por lo que no sería necesario activar dichos módulos.
- **Soporte para cámaras Web**

No es necesario incluir.

Compilamos el kernel como marca el tutorial.

4. Proyecto base en Vivado.

El proyecto base de Vivado, es uno que contiene todas las configuraciones necesarias para el correcto funcionamiento de la tarjeta con sus características y periféricos. Este proyecto, es específico para cada tarjeta. En este tutorial, se nos da ya el proyecto base perfectamente configurado para la tarjeta zybo, pero si quiere hacerlo desde cero, debe ir a la página oficial de la tarjeta [26], descargar el proyecto base y seguir el tutorial de configuración específico. Si fuera para otro tipo de tarjeta, habría que hacerlo igual respectivamente.

A tener en cuenta…

Cada vez que queramos añadir o quitar dispositivos, deberemos acudir al proyecto base, por lo que es conveniente guardarlo para la posteridad.

Todavía no se ha añadido ningún dispositivo, si desea hacerlo en este momento, acceda al punto de como añadir dispositivos y después genere el bitstream.

Si no, puede generar directamente el bitstream, sin dispositivos.

5. (FSBL)First stage bootloader SDK

Guardar ficheros generados.

6. Árbol de dispositivos o DEVICE TREE

Recordar que no hemos reservado memoria para dispositivos.
Comprobar en el fichero "system.dst", que la frecuencia del reloj sea 50000000:

```
&clkc {
        fclk-enable = <0x1>;
        ps-clk-frequency = <50000000>;
};
```

Compilamos el árbol de dispositivo o "device tree".

7. Corriendo Linux por primera vez

No es necesario instalar la aplicación de wifi, ni tampoco la de ejecutar aplicaciones graficas por ssh.

Usuario y contraseña de Linux

Se puede dar cualquier usuario y contraseña pero debido a la implementación, se recomienda utilizar el que se ha aportado:
Usuario : zybo
Contraseña: 123789
PD: Esta contraseña si se cambia deberán cambiarse en varios ficheros que la requieran para realizar sus acciones correctamente.

Instalar Servidor

Para el servidor web es necesario instalar LAMP: instalar LAMP en archlinux[27]

La dirección donde se alojaran los php y los html es "/srv/http/"

Base de datos

Con la base de datos ya instalada, procedemos a crear la base de datos y las tablas correspondientes.
Accedemos desde consola a la base de datos con nuestro usuario y contraseña. En este caso lo hemos llamado root, con contraseña root.
mysql -u root -p
Creamos la base de datos, la llamamos zybo y nos conectamos a ella.
create database zybo;
use zybo;

Creación de las tablas

```
create table dispositivo(
        ID int AUTO_INCREMENT PRIMARY KEY,
        nombre varchar(40) UNIQUE NOT NULL,
        direccion varchar(8) NOT NULL );


create table senal(
        ID int  NOT NULL,
        ID_disp int  REFERENCES dispositivo(ID),
        nombre varchar(40) UNIQUE NOT NULL,
        val varchar(100) DEFAULT 'X',
        sentido varchar(2) DEFAULT 'X',
        actual int DEFAULT 0,
        tipo varchar(50) DEFAULT '0' ,
        PRIMARY KEY(ID, ID_disp) );
```

Instalación de elementos desarrollados.

Copiar a la carpeta "/srv/http/" todos los elementos php, html y la hoja de estilos .css desarrollados que se aportan.

Copiar a una **misma** ruta concreta que se desee la carpeta o el contenido de la carpeta llamada server. En esta se encuentran tanto los scripts Shell de configuración del demonio como los propios módulos implementados en "C" para la ejecución del demonio que actúa de servidor de comunicación interna con la FPGA. **Si se ha utilizado el usuario "zybo, se recomienda instanciar estos archivos como en la muestra, es decir, en la ruta $HOME y dentro de la carpeta aportada llamada "server": $HOME/server/**
Copiar a la "/etc/systemd/system/" los servicios aportados : demonServer.service y red.service.
Ejecutar en linux las siguientes sentencias:

```
sudo systemctl daemon-reload

sudo systemctl enable demonServer.service
sudo systemctl start demonServer.service

sudo systemctl enable red.service
sudo systemctl start red.service
```

Cosas a tener en cuenta

Los html y php deben encontrarse en la misma carpeta.
Scripts de configuración de demonio y los archivos de demonio deben estar en la misma carpeta y de la misma forma que se ha aportado.

Actualizar contraseña usuario

Si la contraseña del super-usuario de linux ha cambiado con respecto a la que se aporta, entonces se deberá actualizar la constante de contraseña en los scripts de configuración del demonio siguientes: **ini.sh**, **red.sh** y **pl.sh**.

Actualizar la ruta del demonio en archivos

Los servicios aportados y algunos php's deben conocer la ruta en la cual se encuentra el demonio y los archivos adjuntos a él. Por lo que si no se ha instanciado los archivos como en la implementación aportada entonces se deberán actualizar a las correspondientes rutas en los archivos: include.php (constante "PATH_UPLOAD"), demonServer.service y red.service.

Listo para usarse

Después de estos pasos ya está en perfecto funcionamiento el servidor en Linux, ya podemos proceder a introducir los archivos de interacción con la FPGA.

B Manual del programador

Añadir dispositivo con VHDL al proyecto base de Vivado

- Vamos al proyecto base creado anteriormente.
- Para que podamos escribir en VHDL, tenemos que indicarlo en las propiedades del proyecto: tools· project settings· Target language · VHDL
- Seguimos el siguiente tutorial de como añadir un dispositivo (viene con un ejemplo)[28].

- Después de añadir el dispositivo, volvemos a realizar el punto 4 (Proyecto base en Vivado) de la instalación de linux, teniendo en cuenta que para crear el HDL wrapper de nuevo, tendremos que borrar el anterior (pulsar encima de él y borrar) y volverlo a crear como se indica.
- Terminar este punto 4, recoger el archivo .bit generado, renombrarlo a "system.bit" .
- Convertir el "system.bit" al formato "bit.bin", formato necesario para poder reconfigurar la FPGA correctamente de forma remota mediante la aplicación proporcionada. Para la conversión utilizamos el programa "bootgen", proporcionado por Vivado. Este se encuentra precisamente en la carpeta de instalación de cualquier versión de Vivado.

Para realizar dicha conversión realizamos los siguientes pasos (se supone que se realiza en Windows":

1. Movemos el recién generado "system.bit" a la carpeta proporcionado con el proyecto "bootgen" junto al archivo "all.bif".
2. Abrimos una consola de Windows y nos desplazamos al contexto de la carpeta mencionada "bootgen".
3. Ejecutamos la siguiente sentencia en consola para realizar la conversión :

RutaDeInstalacionVivado\\bootgen.bat -image all.bif -w -process_bitstream bin

4. Finalizado el proceso, se debe subir el archivo resultante("system.bin.bit") al servidor mediante la página web de subida proporcionada. A esta se puede acceder mediante el acceso al menú de la aplicación.
5. Reconfiguración de FPGA mediante la página de control de los dispositivos pulsando el botón "re-configurar FPGA".
6. Registrar, en caso de que no se haya hecho previamente, la dirección base del mapeo del dispositivo mediante la página de registro.
7. Reprogramar la FPGA mediante el botón que dispone la página control "inter_interaccion.html"

Adicionalmente…

Direccion base: en Vivado, cuando creamos un dispositivo y lo instanciamos en la fpga, se asigna automáticamente una dirección de memoria, con la que se mapea el dispositivo. Esta puede verse y configurarse.

Para acceder a estos elementos debe seguir los siguiente pasos.

1. Abrir el diseño como muestra la figura.

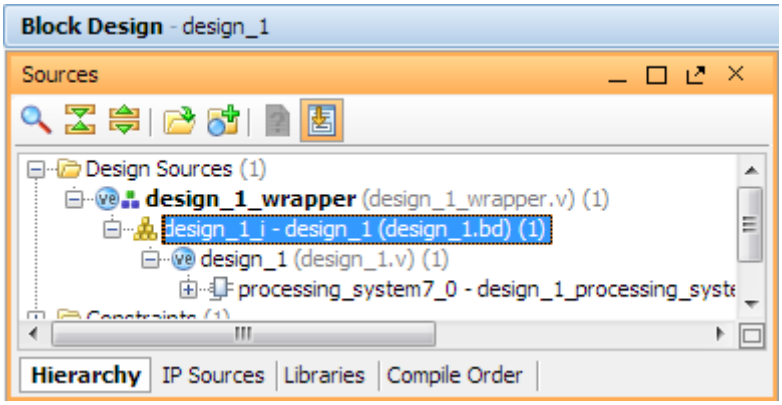


Figura 0-1 Vivado

2. Una vez abierto, debe acceder a la siguiente pestaña:

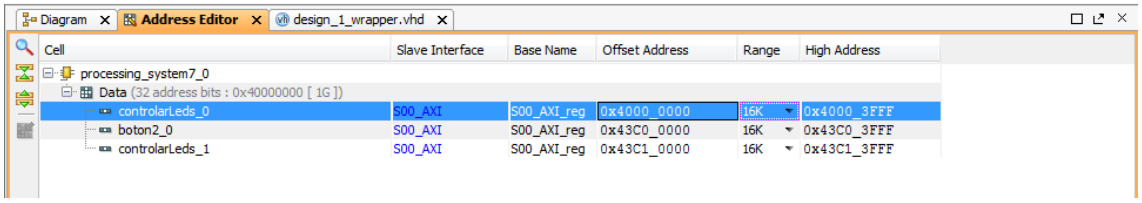


Figura 0-2 Dispositivos Vivado

Como puede verse, se encuentra la dirección del dispositivo el rango. Pulsando encima de ellos puede configurar ambos elementos .

La casilla “offset address” es la dirección base del mapeo con el dispositivo . En la aplicación, la casilla “range” , es decir el rango, es debe ser igual a 16K, por lo que todos los dispositivos deben tener este rango.

¿Cómo accedemos al dispositivo?

Para acceder a los dispositivos se hace mediante un acceso a memoria. Entonces para realizar esta acción hemos implementado una **API,** en lenguaje de programación "C", **que mediante la función "mmap",** se accede a estas direcciones para leer y escribir de las señales de los dispositivos.

Archivos servidor

- **registro.php**

  Este php se encarga de registrar un dispositivo en el sistema mediante los valores que recibe por formulario http. Funciona tanto con GET como con POST.

Variables de entrada:

- disp=nombre: nombre del dispositivo a registrar
- dir=dirección en hexadecimal donde se aloja el dispositivo.
- ran=rango de direcciones del dispositivo en hexadecimal.
- num_sen=x: número de señales a registrar
- regX={I|O}: define el sentido de la señal X. El sentido puede ser "I" u "O". Si hay N señales, se pueden definir reg0, reg1,..., regN-1 sentidos. Por defecto, se suponen que las señales son de entrada("I"), por lo que sólo se necesitan definir los regX que sean de salida(O). La señales deben estar en el orden en el cual se han definido en la fpga(reg0=slv_reg0, reg1=slv_reg1,....., regX = slv_regX)
- tipoX=: define el tipo de señal. Sigue el mismo patrón que el anterior. Por defecto se supone que la señal es hexadecimal, pero puede ser otros tipos.

Ejemplo de registro:
Queremos registrar un dispositivo con 3 entradas, de las cuales las 2 primeras son de entrada y la última es de salida.

http://XXX.XXX.XXX.XXX/registro.php?disp=dispositivo&num_sen=3&reg0=I&reg1=I&reg2=O

como en las señales de entrada puede omitirse el sentido ·

http://XXX.XXX.XXX.XXX/registro.php?disp=dispositivo&num_sen=3&reg2=O

Salida

Mensaje de autenticación y error.

- **borrar.php**

  Este php se encarga de borrar un dispositivo del sistema(o todos) mediante los valores que recibe por formulario http. Funciona tanto con GET como con POST.

Variables de entrada:

Borrar 1 dispositivo determinado:

- disp=nombre: nombre del dispositivo a borrar

Borrar todos los dispositivos: No tiene variables .

Ejemplo:
Borrar dispositivo determinado, borrar dispositivo
http://XXX.XXX.XXX.XXX/borrar.php?disp=dispositivo
Borrar todos los dispositivos:
http://XXX.XXX.XXX.XXX/borrar.php

Salida

Mensaje de autenticación y error.
Nota aclarativa: este módulo php no borra los dispositivos de la FPGA sino que los borra de la base de datos y por tanto la "interfaz" que interactúa con este dispositivo".

- **interaccion.php**

Este php se encarga de controlar las señales de un dispositivo del sistema(o de todos) mediante los valores que recibe por formulario http, además de imprimir en un formato, personalizado, el estado de las señales de los dispositivos. Funciona tanto con GET como con POST.

Variables de entrada:

Borrar un dispositivo determinado:

- **[disp]_[Nsenal]=valor**: indica qué señal de qué dispositivo se debe actualizar. Sólo pueden actualizarse señales de entrada. La variable se forma a modo de "array": **disp** es el nombre del dispositivo al que se quiere acceder, y Nsenal es la posición que ocupa la señal dentro del dispositivo (debe ir acorde con lo registrado). El valor debe ir acorde al tipo de señal que es(hexa, coma flotante…). Finalmente decir, que se pueden actualizar tantas señales como registros de estas haya, o por lo contrario, no actualizar ninguna.
- **modo**={json|interf}: es el formato en el cuál, el servidor devuelve el estado de los dispositivos del sistema. Por defecto, lo devuelve en formato json, por lo que sólo se pone cuando se quiera en formato interfaz.

Ejemplo:
Se quiere actualizar la primera señal del dispositivo X, y la segunda señal del dispositivo Y que devuelva en modo interfaz.
http://XXX.XXX.XXX.XXX/interaccion.php?X_0=valor1&Y_1=valor2 &mod=json
sólo se quiere la impresión de los dispositivos:
http://XXX.XXX.XXX.XXX/interaccion.php

Salida:

El estado de los dispositivos en el formato dado o mensaje de error.

- **uploader.php**

módulo php que recibe el archivo de reprogramación de la FPGA. El único nombre y formato que permite que se suba es "system.bit.bin". El tamaño máximo es de 10 MB. Si no cumple cualquiera de estas premisas se devuelve un error. Si cumple todas las premisas mueve el archivo de configuración a la carpeta indicada por la constante "**PATH_UPLOAD"** (alojada en "includes.php").

- **includes.php**

  Este php no tiene un uso directo por el usuario, sólo cuenta con "primitivas" para acceder a la base de datos, funciones auxiliares y definición de constantes, que por otro lado, son utilizados por el resto de phps.

Algunas definiciones importantes

Sólo puede haber una referencia a la base de datos a la vez.

Datos de acceso a la base de datos:

DB_IP: dirección IP donde se aloja la base de datos, por defecto localhost.
DB_NAME: nombre de la base de datos a la que accedemos.
DB_USER: usuario de la base de datos.
DB_PASS: contraseña de la base de datos.
TIPO_BD: tipo de base datos, por defecto mysql.
PATH_UPLOAD: ruta completa a la carpeta donde se encuentra el demonio PS-FPGA.

Primitivas de acceso a la base de datos:

function conectarDB() : Conecta a la base de datos devuelve la referencia a la descriptor de base de datos

function insertarDB($consulta): inserta una tupla en base de datos, $consulta es la consulta a ejecutar

function cerrarDB():cierrar conexion con la db
function crearUpdatefunción auxiliar que utiliza "interaccion.php", y que encapsula todo el procedimiento que realiza este módulo.
function insertarSenales: función auxiliar que utiliza "interaccion.php", y que encapsula todo el procedimiento que realiza este módulo.
function imprimirJSON(): función que se encarga de imprimir el estado de los dispositivos en formato Json.
function imprimirDebugBR($mes): función auxiliar de impresión de mensajes a modo de debug.

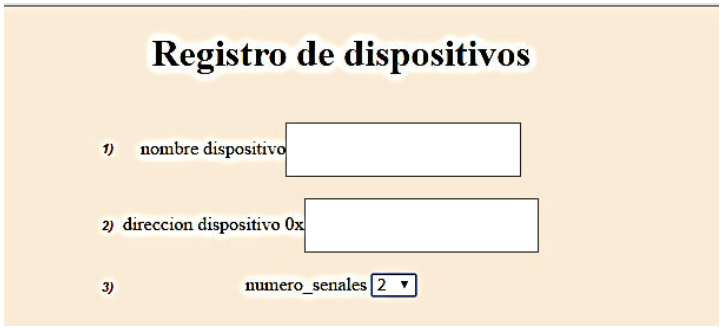
Archivos interfaz de usuario

- **Index.html**

  Menú de navegación entre las funciones de control del usuario: registro de dispositivos, interacción con dispositivos registrado o eliminación de dispositivos.

- **Inter_registro.html**

  Archivo html + javascript por el cual interactúa de manera intuitiva con el archivo registro.php, y con el cual se registra dispositivo.

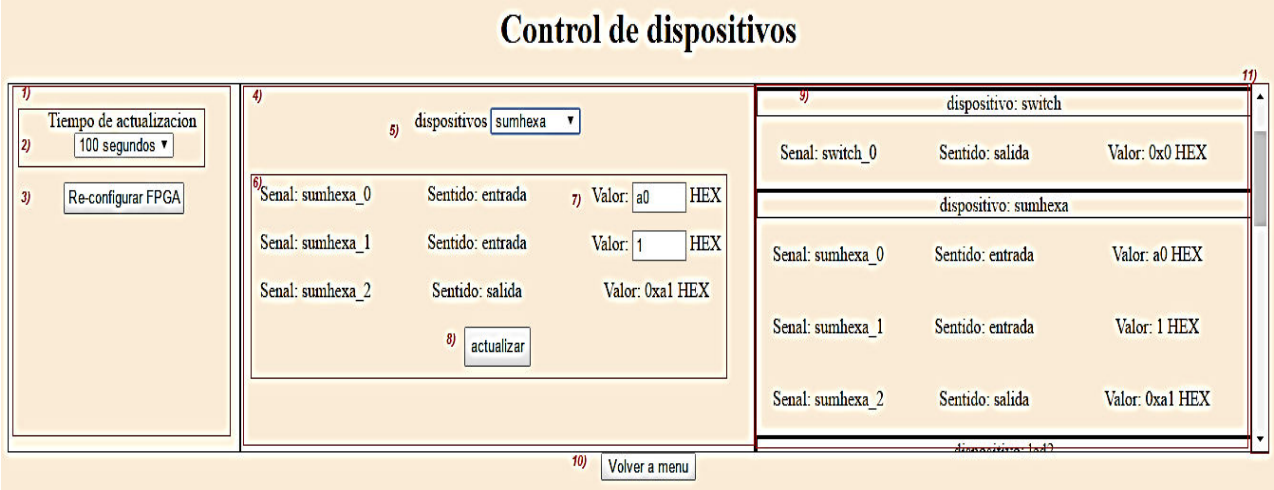


Se presenta el formulario de registro.

1. Nombre del representativo del dispositivo dentro del sistema.
2. Dirección de memoria de mapeo con el dispositivo.
3. Selector de número de señales que tiene el dispositivo. Mínimo una. Máximo 10.
4. Selector de tipo de representación de los datos con los que trabaja la señal. Ahora sólo permite Hexadecimal y Sfixed(coma fija configurable).
5. Selector del sentido de la señal. Entrada o salida.
6. Botón de envío de formulario.
7. Botón de vuelta al menú.
8. Barra de desplazamiento dentro del formulario.

- **Inter_interaccion.html**

  Este archivo html + javascript, trae la información y estado de los dispositivos registrados y alojados en la FPGA. Mediante una simple interfaz, se puede controlar los dispositivos y sus señales de forma fácil. Adicionalmente se puede: mediante un desplegable elegir el tiempo de actualización de los dispositivos y mediante un botón reprogramar la FPGA en caso de que se haya subido un archivo de reprogramación anteriormente.

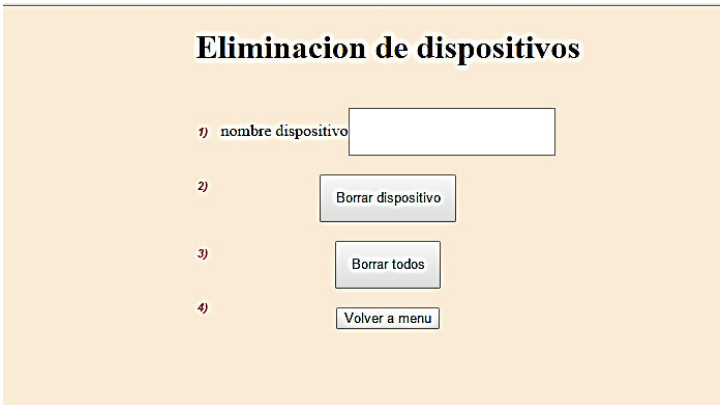


1. Cuadro de opciones
2. Selector de tiempo de actualización.
3. Botón de reconfiguración de FPGA con el archivo de configuración previamente subido.

4. Cuadro de formulario y control de dispositivos.
5. Selector de dispositivos.
6. Formulario del dispositivo elegido con los datos de sus señales.
7. Cuadro de texto que aparece en la señales de entrada y sirve para darle valores a esta.
8. Botón de envío de formulario que actualizará las señales re evaluadas.
9. Cuadro de información de todos los dispositivos registrado y alojados en la fpga.
10. Botón de salida de la página hacia el menú.
11. Barra de desplazamiento para navegación en el cuadro de información de los dispositivos.

- **Inter_borrar.html**

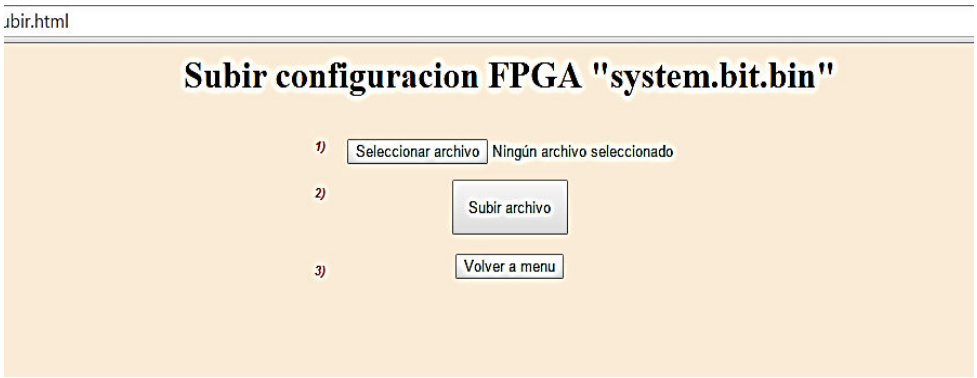
  Este archivo html +javascript, permite borrar o un dispositivo concreto identificado por su nombre, o todos los dispositivos directamente.



1. Nombre del dispositivo especifico a eliminar (nombre representativa con el cual se registró este).
2. Botón para enviar formulario y quitar su registro de la base de datos.
3. Envía formulario para borrar todos los dispositivos que están registrados en la base de datos del sistema.
4. Botón para volver al menú.

- **subir.html**

  Esta página permite mediante un formulario intuitivo subir un archivo de reprogramación de la FPGA.



1. Elemento de formulario que permite elegir archivo que se subirá al servidor a la al pulsar el botón de envío.
2. Botón de envío de formulario que subirá el archivo.
3. Botón para volver al menú.

Proceso Demonio Interfaz PS-FPGA en C

- **mysql_mod.c**

  Contiene las funciones relevantes para acceder a base de datos mysql. Sólo puede haber una referencia a la base de datos. También se le ha añadido una pila de consultas(consultas sin resultado). Estas se almacenan en una array y se ejecutan de una vez.

  int conectarDB(): conecta a la base de datos, devuelve código de error OK o ERR

  void cerrarRes(MYSQL_RES * res) : cierra un resultado de una consulta de forma segura.
  int consultaEditarDB(char * query) : realiza una consulta sin resultado(update o delete)

  int insertarSQLPila(char * consulta): inserta una consulta(update o delete) en la pila
  int ejecutarPila(): ejecuta la pila de consultas
  MYSQL_RES * consultaDB(char * query): realiza una consulta con resultado a la base de datos.
  MYSQL_ROW getResultado(MYSQL_RES * res): obtiene el resultado de realizar una consulta. Res es el puntero devuelto por la función consultaDB
  int iniTransaccionDB(): inicia una transacción en la base de datos.
  int rollBackTransaccionDB(): realiza un rollback de la transacción.
  int commitTransaccionDB(): realiza un commit de la transacción.
  void closeDB(): cierra toda la base de datos.

- **mysql_mod.h**

  Contiene definiciones de la base de datos prototipo de las funciones de acceso a la base de datos.
  Código de error : **OK y ERR**.
  Credenciales de acceso a la base de datos:
  SERVER : IP de la base de datos (por defecto localhost).
  USER: usuario de acceso a la base de datos
  PASSWORD: contraseña del usuario de la base de datos.
  DATABASE: nombre de la base de datos a la que se desea acceder.

  IMPORTANTE: este diseño sólo conecta con base de datos tipo mysql.

- **dispositivo.c**

  Contiene las funciones de acceso a los dispositivos alojados en la fpga(previamente definidos en el device tree con el procedimiento descrito anteriormente.

Variables globales

Estas son sólo utilizadas por las funciones apiDisp, iniApiDisp(int id_disp) y apiCerrarDisp(). Las demás funciones son independientes.

```
int disp= 0;
volatile unsigned *desc = NULL;
int aux_id = -1;
int debug = 1;//si = 1 activa debug
```

Funciones genéricas de dispositivos:

int inicioDisp(char * dispositivo): abre el dispositivo y retorna el identificador del descriptor. Puede devolver código de error ERR.

volatile unsigned * getDispositivo(int id_disp): obtiene el puntero a memoria del dispositivo dado el id del descriptor. Este id se obtiene al abrir un dispositivo con inicioDisp. Puede devolver ERR.

int escribirDisp(volatile unsigned * desc, unsigned int * buffer, int tamano): escribe dentro de un dispositivo.

desc: es el puntero al fichero(puntero devuelto por getDispositivo.
buffer: cadena de datos a escribir en el dispositivo.
tamano: número de registros a escribir.
Puede devolver ERR.

int leerDisp(volatile unsigned * desc, unsigned int * buffer, int tamano): lee de un dispositivo.

desc: es el puntero al fichero(puntero devuelto por getDispositivo.
buffer: buffer donde se guardara los registros leidos.
tamano: número de registros a leer.
Puede devolver ERR.

void cerrarMemoria(volatile unsigned * desc): cierra el el puntero del dispositivo.

void cerrarDispositivo(int disp): cierra el dispositivo abierto, disp Es el id al dispositivo.

Funciones específicas para el sistema:

int inicioUIO(unsigned int id_disp): realiza una encapsulacion de la apertura de un dispositivo UIO(tener en cuenta que empieza en uio0).

int iniApiDisp(int id_disp): función que encapsula la llamada de para iniciar nuestro dispositivo de forma específica(uio) para nuestro sistema. Id_disp =(0,1,2….)

int apiDisp(int modo,char * identificador,unsigned int id_disp, unsigned int registro, unsigned int * dato)
Función que encapsula el algoritmo específico, sobre la forma de escribir o leer, en/de un dispositivo generado en el actual sistema. (En el futuro, esta función puede cambiar su funcionamiento interior. En principio será invisible para el usuario).
int modo: modo de apertura : READ_D WRITE_D.
char * identificador : no se usa.
int id_disp: id del dispositivo en el device tree, de la forma uioX (ejemplo
id_disp = 1 => se abrirá el dispositivo definido en el device tre como "uio1")
int registro: registro o señal a escribir/leer dentro del dispositivo
output ERR o OK
void apiCerrarDisp(): Cierra el dispositivo específico
void imprimir(char * cadena): imprime a modo debug, si este esta activado.

- **dispositivo.h**

  Este módulo, contiene los prototipos de las funciones de acceso a los dispositivos de la fpga.
  Define códigos de error: OK, ERR
  PATH_MSIZE: esta constante guarda la ruta al archivo que guarda el rango de direcciones de cada dispositivo registrado en el device tree. Esta ruta depende de la plataforma en la que estemos, en archlinux, es la que está definida. En caso de que no se defina se tomará por defecto un rango de ***0x1000,*** *definido en la constante*
  MAP_SIZE , también en este módulo**.**

- **daemon.c**

  int daemon_open: inicia el demonio del programa.

  void daemon_close : función manejador de señales de salida para que el programa salga de su estado de bucle infinito y salga del programa.

- **daemon.h**

  Contiene los prototipos de las funciones.

- **Log.c**

  Contiene funciones que permiten loguear información o errores, con formato de fecha y hora. El nombre del fichero del log ese compone con log +[_fecha]

  int loggear(char * linea): loguea en modo información el mensaje línea.
  Esta funcion depende de la variable global flag_log , bandera que permite configurar si hay logueo o no , logueo =1, entonces activo.

  int loggearERR(char * linea): igual que el anterior pero para errores. Esta funcion no depende de la variable flag_log.

- **log.h**

  Contiene prototipos de funciones para las funciones de logueo, además tiene definiciones de constantes.
  F_NAM: parte estática del nombre del log
  INFOR_MSG: parte estática del mensaje de información de log.
  ERROR_MSG: igual que el anterior pero para error.

- **mensajes.h**

  Contiene definiciones de mensajes usuario, normalmente para el log.

- **conversor.c**

  Contiene las funciones necesarias y auxiliares para llevar a cabo la conversión de tipos para formatearlos adecuadamente al tipo de señal que espera recibir las señales de los dispositivos. El sistema suele convertir un numero en notación convencional como puede ser en hexadecimal o número en decimales en forma de cadena de caracteres, a una notación descrita para el funcionamiento de los dispositivos, ya que estos, trabajan con bits.

  Actualmente sólo hay 2 tipos definidos:

  Hexadecimal: tipo por defecto. Sin argumentos. **Tipo "0".**

  Sfixed: tipo coma fija. Este tipo utiliza el bit de mayor significado como signo, y el resto es para definir el dato, con mayor o menor precisión dependiendo de los bits que se dediquen a la parte entera/decimal. La suma del número de bits dedicados a la parte entera, más los dedicados a la parte decimal, más el bit dedicado al signo, deben ser, obligatoriamente, igual al tamaño de palabra. Debido a lo mencionado anteriormente, para poder llevar a cabo su conversión y por decisiones de diseño, este tipo debe llevar con él un argumento, el número de bits dedicados a la parte decimal. El dominio de este argumento es de 0 a (**BitsTamañoDePalabra** – 1). **Tipo "1".**
  Las siguientes funciones definen el proceso de conversión:
  int convertNumToString(int * dato_in , char * args, char * dato_out):
  Función que encapsula el procedimiento de conversión de cualquier tipo definido, de número a cadena de caracteres.

  Dato_in: es la dirección al número a convertir.
  args: cadena de caracteres en la cual se definen el tipo y argumentos para la conversión del dato, estos pueden venir en el siguiente formato
  Tipo[**_argumento**]*. **Tipo** siempre es obligatorio y el argumento depende del formato,

  Ejemplo de entrada: "0" // sería el tipo hexadecimal, no tiene argumentos.
  Ejemplo de entrada: "1_20" // tipo SFIXED, que como está definido, tiene un argumento que define el número de bit dedicados a los decimales del dato a convertir. En este ejemplo 20 bits para describir los decimales.

  dato_out: dato convertido en forma de cadena de caracteres.

int convertStringToNum(char * dato_in, char * args,int * dato_out): realiza el procedimiento contrario a la anterior función. Se estructura es igual a la anterior, pero con los datos en entrada y salida al contrario.

int convertStringToSFixed(char * dato_in, int n_b_entera, int n_b_decimal ,int * dato_out): función que convierte un numero real en forma de cadena de caracteres en uno en formato coma fija.

dato_in: dato de entrada en cadena de caracteres.
n_b_entera: número de bits para la parte entera**.**
n_b_decimal: número de bits para la parte decimal.
dato_out: dato convertido.

n_b_entera + n_b_decimal == Tamaño de palabra – 1. En este caso definido en la constante **N_BITS,** como 32 bits.
int convertSFixedToString(int * dato_in, int n_b_entera, int n_b_decimal , char * dato_out): realiza el procedimiento contrario a la anterior función. Misma estructura pero con tipos de datos de entrada/salida al contrario.

En caso de querer definir más tipos

1. Definir procedimiento específico de conversión/des-conversión del dato.
2. Añadir entrada para el tipo en las funciones **convertStringToNum y convertNumToString**(definir número de argumentos y preparación para el procedimiento especifico de conversión) y llamar a la función definida en el paso 1

- **main.c**

Módulo que une todos los demás módulos en un sólo algoritmo. Este además está a la escucha de que exista un fichero de configuración de la FPGA con el "bitstream.bit.bin" dentro de la su misma carpeta. Si existe lo ejecuta, lo borra y reinicia el servidor.